VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SILNÁ KRYPTOGRAFIE NA ČIPOVÝCH KARTÁCH
STRONG CRYPTOGRAPHY ON SMART CARDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Konečný**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Lukáš Malina, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Jakub Konečný ***ID:*** 154770
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Silná kryptografie na čipových kartách

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s moderními programovatelnými čipovými kartami (JAVA, MULTOS, .NET a Basic). Jednotlivé typy karet analyzujte a porovnejte z hlediska nabízených kryptografických funkcí a míry zabezpečení. Otestujte a změřte dobu výpočtu u vybraných kryptografických metod na zvolených čipových kartách. Na základě výkonnostního zhodnocení zvolte vhodnou platformu pro návrh a implementaci bezpečného autentizačního protokolu s ustanovením klíčů (např. protokol PACE). Navržený protokol naprogramujte na čipovou kartu (strana uživatele) i na PC (strana ověřovatele se čtečkou karet). Změřte paměťovou a výpočetní náročnost implementovaného protokolu a proveďte případnou optimalizaci řešení i implementace.

DOPORUČENÁ LITERATURA:

[1] RANKL, Wolfgang. Smart Card Applications: Design Models for Using and Programming Smart Cards. 2007. 217 s. ISBN 9780470058824.

[2] MENEZES, Alfred, VAN OORSCHOT, Paul, VANSTONE, Scott. Handbook of applied cryptography. Boca Raton : CRC Press, 1997. 780 s. ISBN 0849385237.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: Ing. Lukáš Malina, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Práce je zaměřena na kryptografii na čipových kartách. Popisuje čipové karty, jak z hardwarového, tak softwarového hlediska. Srovnává známé operační systémy používané na čipových kartách. Vybrané systémy podrobněji zkoumá. Dále jsou v práci představeny vybrané protokoly pro autentizované ustanovení klíče. Protokol PACE je podrobněji představen a je navrhnuta jeho implementace. Následně je implementován na čipových kartách Basic card. Implementace je blíže analyzována a v závěru jsou navržena možná vylepšení.

KLÍČOVÁ SLOVA

čipové karty, Java karty, Basic karty, MultOS, .NET karty, protokol PACE

ABSTRACT

The diploma thesis is focused on cryptography with smart cards. It describes smart cards from hardware and software side. The thesis compares well known operating systems used on smart cards. Specified operating systems are widely analyzed. There is introduction to password authenticated key establishment protocols. PACE protocol is described in details together with implementation proposal. The implementation on the Basic card platform follows. In the end there is the time analysis of the implementation and further improvements are suggested.

KEYWORDS

Smart cards, Java cards, Basic cards, MultOS, .NET cards, PACE protocol

KONEČNÝ, J. *Silná kryptografie na čipových kartách*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2017. 50 s. Diplomová práce. Vedoucí práce: ing. Lukáš Malina, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Silná kryptografie na čipových kartách jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Lukáši Malinovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Stálé více lidí používá platební karty. Rozšiřují se elektronické doklady, jako jsou pasy a občanské průkazy. Velká část firem a škol používá přístupové systémy založené na čipových kartách. Z těchto důvodů je potřeba, aby komunikace mezi kartou a terminálem byla dostatečně zabezpečená.

Úvod práce obsahuje představení čipových karet, popis jednotlivých typů, strukturu dat a způsoby komunikace s terminálem. Dále práce srovnává operační systémy používané na kartách a vybrané systémy jsou blíže analyzovány.

Druhá část práce představuje protokoly pro ustanovení šifrovacího klíče na základě předsdíleného hesla, PAKE protokoly. Mezi tyto protokoly patří i protokol PACE, který je v práci podrobně představen, včetně návrhu implementace.

Následně je protokol PACE implementován na čipových kartách Basic card, část ověřovatele je naprogramována v jazyce Java. Na provedené implementaci bylo provedeno několik měření pro různé délky klíčů a různé eliptické křivky. V závěru práce je uvedena bezpečnostní analýza protokolu a návrh na možné zlepšení implementace.

1 ZÁKLADNÍ POPIS ČIPOVÝCH KARET

Kapitola se zabývá možnostmi využití čipových karet a dělením karet dle různých kritérií. V další části následují způsoby zabezpečení karet a detailnější pohled na jejich hardware.

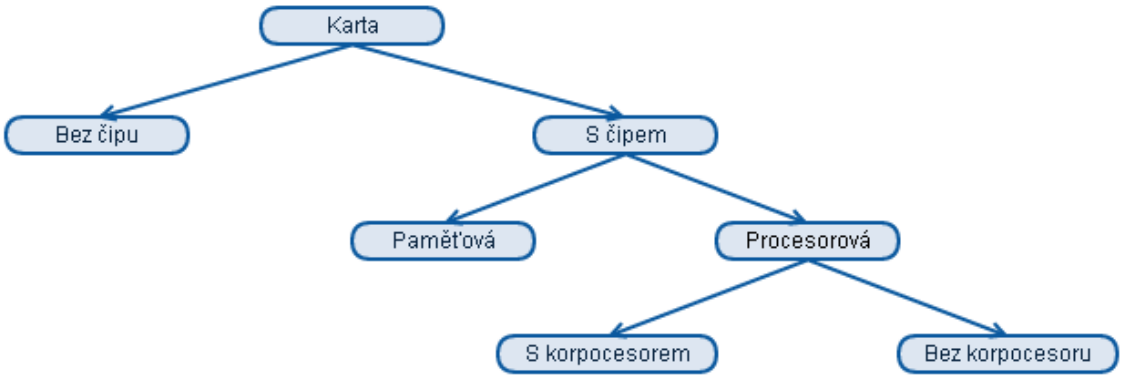
1.1 Využití čipových karet

S čipovými kartami se setkáváme prakticky denně, i když si to většina lidí možná ani neuvědomuje. Nejznámější a jedno z nejrozšířenějších využití je ve finančnictví – platební karty. Tato oblast vyžaduje díky své povaze velmi dobré zabezpečení karet, jak ze softwarového, tak z hardwarového hlediska.

Další oblastí jsou komunikační systémy, obvykle se jedná o SIM kartu v mobilním telefonu nebo přístupovou kartu pro televizní vysílání (kabelové DVB-C nebo satelitní DVB-S). Ve školách a firmách se využívají paměťové nebo čipové karty k řízení přístupu a autentizaci studentů/zaměstnanců. Čipy se pomalu dostávají i do karet jako jsou občanský průkaz, řidičské oprávnění nebo karty na městskou hromadnou dopravu. Supermarkety používají věrnostní karty k optimalizaci rozložení produktů a sledování poptávky.

1.2 Rozdělení karet dle hardwaru

Dle hardwaru rozdělujeme karty na karty s čipem (čipové karty) a karty bez čipu. Karty bez čipu mohou obsahovat pouze embosování, magnetický pásek, apod. Čipové karty dále můžeme rozdělit na paměťové a procesorové (viz Obr. 1).



Obr. 1: Rozdělení karet dle hardwaru

1.2.1 Paměťové karty

Paměťové karty neobsahují procesor a mohou pouze uchovávat data. Obsahují paměť ROM, která obsahuje informace o paměťovém čipu, a paměť EEPROM, která je přepisovatelná. V EEPROM je uloženo identifikační číslo čipu a další volitelné informace. Přístup k této paměti může být chráněn PIN kódem.

1.2.2 Procesorové karty

Procesorové karty obsahují procesor (CPU), který umožňuje spouštění jednoduchých aplikací. Dále zde nesmí chybět paměti ROM, EEPROM a RAM. V EEPROM může být uložený operační systém karty, RAM slouží k ukládání dočasných dat, nutných pro běh aplikace.

Procesor může být doplněn o koprocesor, jenž mívá na starost složité matematické výpočty, např. při asymetrickém šifrování RSA nebo kryptografii využívající eliptických křivek (ECC). Proto je často označován jako kryptoprocesor.

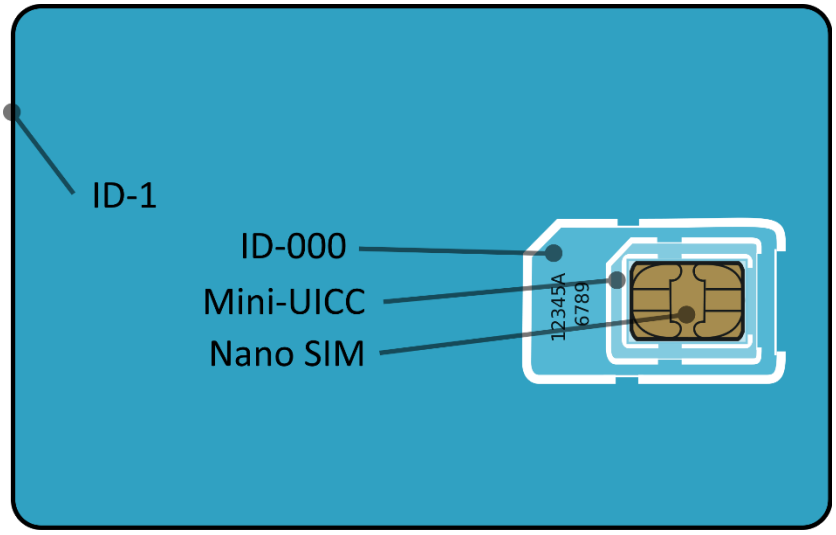
1.3 Rozdělení karet dle způsobu komunikace se čtečkou

Dle způsobu komunikace se čtečkou karty dělíme na kontaktní a bezkontaktní. Kontaktní karty komunikují se čtečkou pomocí pozlacených kontaktů, které se přímo dotýkají čtečky.

Bezkontaktní karty využívají bezdrátového přenosu dat. Jejich komunikace je definována standardem ISO/IEC 14443. Spojení je umožněno až na vzdálenost 10cm. Karty jsou napájené ze čtečky pomocí elektromagnetické indukce.

1.4 Rozměry karet

Všechny karty mají společnou tloušťku 0.76mm. Šířka a výška se ovšem liší a existuje mnoho více či méně rozšířených standardů. Jelikož se dotykové karty vkládají do čtečky, je potřeba, aby měly, kvůli vzájemné kompatibilitě, stejné rozměry a proto se dnes u většiny platebních a přístupových karet používá formát ID-1. V mobilních telefonech se výrobci snaží o čím dál větší miniaturizaci, čímž se dostali od Mini SIM karty (ID-000), přes Micro SIM (Mini-UICC) až po Nano SIM (viz Obr. 2).



Obr. 2: Rozměry karet

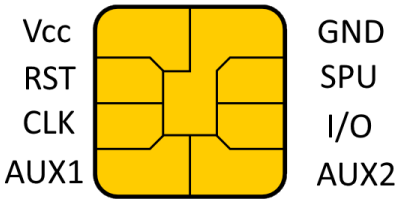
Přesné rozměry jsou v Tab. 1.

Tab. 1: Přesné rozměry karet

Formát karty	Šířka (mm)	Výška (mm)
ID-1	85.6	54
ID-000	25	15
Mini-UICC	15	12
Nano SIM	12.3	8.8

1.5 Čip modul

Modul je vlastně ochranné pouzdro obsahující samotný mikrokontrolér s osmi vývody (viz Obr. 3). Běžně se ovšem používá pouze 5 vývodů a zbylé 3 jsou rezervované pro budoucí využití.



Obr. 3: Vývody čipu

- V_{cc} – napájecí napětí
- GND – uzemnění
- RST – reset

- SPU – standartní nebo proprietární použití (standard or proprietary use)
- CLK – hodinový signál
- I/O – vstup/výstup
- AUX1, AUX2 – nepoužívané pomocné (auxiliary) vývody

1.6 Přenos dat

Komunikace mezi čipovou kartou a terminálem probíhá vždy v režimu half duplex, tzn. vždy probíhá komunikace pouze jedním směrem, ze čtečky ke kartě nebo naopak. Aby bylo možné předejít kolizím, inicializuje komunikaci vždy čtečka. Čtečka je tedy master (posílá dotazy) a karta je slave (odpovídá).

Po vložení karty do čtečky vyšle čtečka příkaz reset a karta odpoví signálem ATR (answer to reset), který obsahuje informace o přenosových protokolech a přenosových rychlostech podporovaných kartou. Poté volitelně následuje sekvence příkazů PPS (protocol parameter selection). Tuto komunikaci iniciuje terminál v případě, že chce změnit parametry přenosu získané z ATR 0.

1.6.1 Transportní vrstva

Kontaktní karty používají ve většině případů na transportní vrstvě protokoly T=0 a T=1. Občas se lze setkat i s protokolem USB, případně u bezkontaktních karet s ISO/IEC 14 443. Datová jednotka se nazývá TPDU (Transport Protocol Data Unit).

Protokol T=0 je starší a jednodušší, je bajtově orientovaný a nepodporuje všechny typy APDU (application protocol data unit) zpráv. Používá se například u SIM karet. Protokol T=1 je blokově orientovaný a podstatně složitější než T=0. Jeho velkou výhodou je možnost detekce a znovu odeslání chybových bloků. Používá se například u platebních karet.

1.6.2 Aplikační vrstva

Datová jednotka aplikační vrstvy se nazývá APDU. APDU se dělí na příkaz posílaný terminálem – APDU command a odpověď posílanou kartou – APDU response.

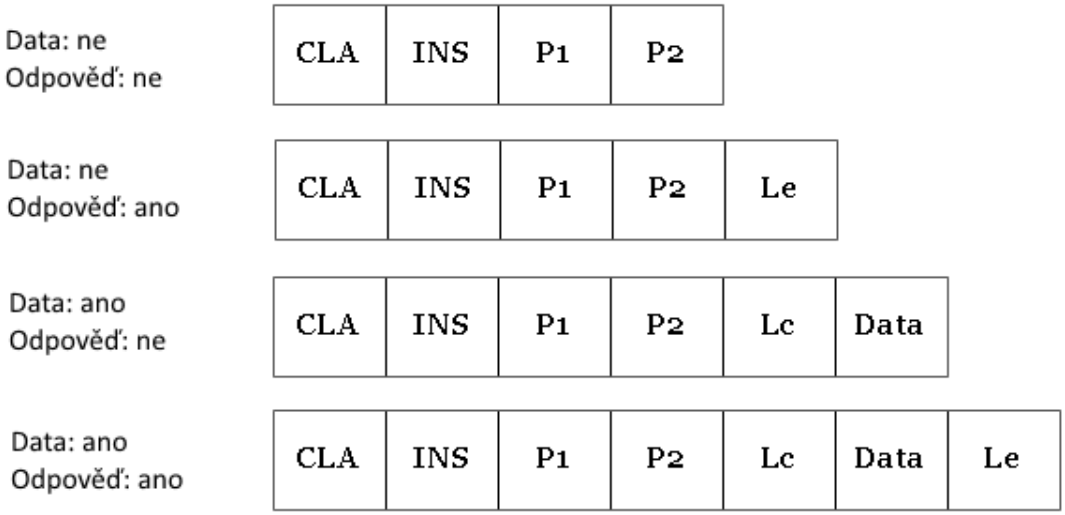
APDU command se skládá z hlavičky a těla. Hlavička obsahuje 4 bajty:

- CLA – instrukční třída,
- INS – instrukční kód,
- P1 – parametr příkazu,
- P2 – parametr příkazu.

Tělo příkazu se skládá ze tří částí:

- L_c – délka posílaných dat,
- Data,
- L_e – délka očekávané odpovědi.

APDU command může být posílán ve čtyřech různých variantách, kdy povinná je vždy pouze hlavička (viz Obr. 4).



Obr. 4: Čtyři varianty příkazu APDU

APDU response se skládá ze tří částí:

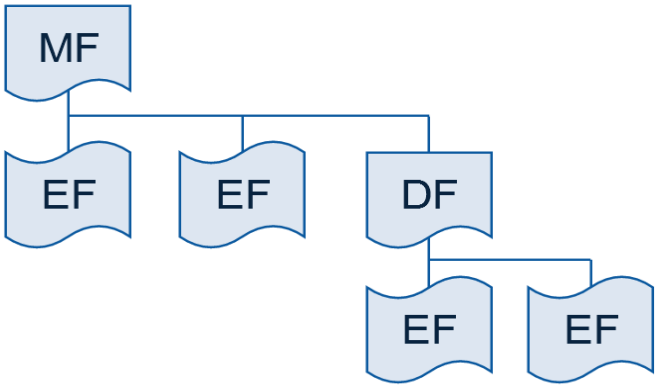
- Data,
- SW1
- SW2

SW1 a SW2 udávají status odpovědi v hexadecimální soustavě. Např. 90 00 představuje úspěšnou odpověď.

1.7 Struktura souborů

Čipové karty mají specifickou strukturu složek a souborů nezávislou na použitém operačním systému. Kořenová složka se nazývá MF (master file). Master file se dělí na podsložky, DF (dedicated file), které je teoreticky možné větvit donekonečna. Většinou se používají 3 nebo 4 úrovně a operační systém obvykle neumožňuje více než 8 úrovní. Existuje i speciální složka ADF (application dedicated file), která se vztahuje k určité aplikaci a může být umístěna i mimo kořenovou složku.

Data aplikací a operačního systému jsou ukládána v EF (elementary file). Tyto soubory musí být vždy ve složce DF nebo MF. Dělí se na dva typy: pracovní EF a interní EF. Pracovní EF ukládají aplikační data a jsou přístupná přes APDU příkazy. Interní EF se používají pro ukládání dat, která nemají být přístupná terminálu, např. šifrovací klíče (viz Obr. 5).



Obr. 5: Struktura dat na čipové kartě

2 SROVNÁNÍ OPERAČNÍCH SYSTÉMŮ NA ČIPOVÝCH KARTÁCH

Možnosti čipových karet jsou mnohem více závislé na použitém operačním systému (OS), než na samotném hardwaru karty. Díky operačnímu systému je možné karty využívat pro široké spektrum činností.

Operační systém je uložen v paměti ROM. Většina dostupných systémů je založená na skupině standardů ISO/IEC 7816.

Díky OS mohou uživatelé a vývojáři psát vlastní aplikace pro čipové karty a mohou používat jím známé programovací jazyky, jako např. Java, C# nebo Basic. Interpreter v operačním systému dokáže aplikaci přeložit do strojového kódu pro příslušný procesor. Vývojář musí kromě aplikace, která běží na kartě, vytvořit i klientskou aplikaci pro terminál komunikující s kartou.

Mezi nejpoužívanější systémy patří Java Card, Basic Card, .NET a MULTOS.

2.1 Java Card

První Java karty byly představeny v roce 1996 firmou Schlumberger Limited. Jejich cílem bylo přinést multiplatformní programovací jazyk Java i do prostředí čipových karet. Na začátku bylo definování Java card API, obsahující funkce potřebné pro práci s kartou a velmi důležité jsou zde i kryptografické funkce. Dále se musel upravit virtuální stroj pro běh Java aplikací (JVM), čímž vzniká Java card virtual machine (JCVM). Aplikace pro Java karty se nazývají applety.

2.1.1 JCRE (Java Card Runtime Environment)

Prostředí JCRE se stará o instalaci apletů na kartu a poté i o jejich běh. Součástí JCRE je jak virtuální stroj pro běh aplikací, tak i framework a API, které obsahují třídy a balíčky následně využívané při spouštění appletů na kartě (viz Obr. 6).

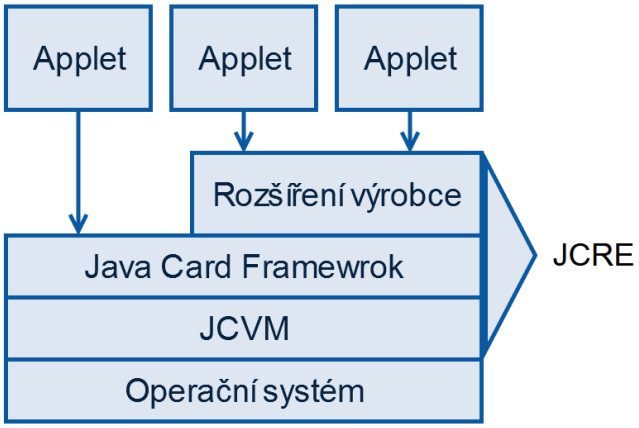
Životní cyklus appletu začíná zavoláním JCRE metody *Applet.register* a končí až smazáním appletu nástrojem *Applet Deletition Manager*. Při instalaci appletu na kartu je nutné vytvořit jeho instanci metodou *install*. Tato metoda zavolá konstruktor appletu, např. vytvoří objekty v trvalé EEPROM paměti, pokud tam applet plánuje ukládat data potřebná pro běh. Nakonec se zavolá *Applet.register()*, pokud registrace proběhne v pořádku, je applet bez problémů nainstalován. V případě, že registrace vytvoří výjimku nebo metodou *install* vůbec nebyla zavolána je instalace považována za neúspěšnou.

Pokud chceme applet na kartě spustit, provede to JCRE pomocí metody *select*, která se spouští, pokud příchozí APDU obsahuje v datové části identifikační číslo appletu (AID). Nejprve je nutné provést *deselect* metodu u předchozího vybraného appletu a následně můžeme vybrat aktuální applet, přičemž applet může svůj výběr odmítnout vrácením hodnoty *false*.

Z předchozího vyplývá, že JCRE se stará o zpracování všech APDU. Přijatý APDU je předán metodě *process* a dle obsahu APDU je vykonána potřebná úloha. JCRE automaticky na konec odpovědi přidá 2 bajty 9000_{16}, jestliže vše proběhlo bez chyb. Jakmile applet vrátí výjimku, je její kód odeslán s odpovědí místo bajtu 9000_{16} [2].

Pokud dojde ke ztrátě napájení karty, JCRE se postará o následující činnosti:

- přechodná data resetuje do původního stavu,
- pokud byla vykonávána transakce, je zrušena,
- vybraný applet je odebrán,
- je vybrán applet, nastavený jako výchozí.



Obr. 6: Čipová karta - Java Card

2.1.2 Java card API

Java card API velkou měrou určuje, co daná karta dokáže. Při koupi karty je verze API velmi důležitý parametr. Když karta podporuje např. API verze 2.2.1, nelze využívat funkce novějších API. Java karty vyrábí mnoho výrobců a i přes to, že výrobce udává konkrétní verzi podporovaného API, neznamená to, že karta zvládne všechny jeho funkce. Ke zjištění všech dostupných kryptografických funkcí dané karty existuje software českého vývojáře Petra Švendy JCAlgTest. Kromě verze API software vyzkouší všechny kryptografické funkce a vypíše seznam funkcí kartou podporovaných, zároveň dokáže i změřit čas vykonávání funkcí s různými vstupními parametry.

Zásadní verze API a nové funkce v nich přidané:

- verze 2.1 (1999),
- verze 2.1.1 (2000),
  - přidána podpora pro RSA bez zarovnání (padding),
- verze 2.2 (2002),
  - přidána podpora pro šifrování AES a ECC klíčů, CRC algoritmy, ECDH algoritmus, nové délky klíčů pro AES a RSA,
- verze 2.2.1 (2003),
  - změny ukládání parametrů u eliptických křivek, změna ukládání inicializačního vektoru u AES šifry,
- verze 2.2.2 (2006),
  - přidány algoritmy SHA-256, SHA-384, SHA-512, HMAC, SEED,
- verze 3.0.1 (2009),
  - přidána podpora SHA-2 pro všechny algoritmy digitálního podpisu,
- verze 3.0.4 (2011),
  - přidány algoritmy DES MAC4 a DES MAC8, vylepšená metoda pro generaci klíčů pro jednotlivé šifry,
- verze 3.0.5 (2015),
  - přidány algoritmy AES CMAC, SHA3, AEAD-CCM, AEAD-GCM a podpora pro PACE [3].

Od verze API 3 byly vytvořeny dvě nezávislé větve nazvané Classic Edition a Connected Edition. Classic Edition rozšiřuje předchozí verze API, zatímco Connected Edition představuje karty schopné komunikovat v IP sítích. Tyto karty se mohou chovat jako síťový uzel, poskytovat bezpečnostní služby síti a požadovat přístup k síťovým prostředkům. Karty jsou také přístupné pomocí HTTP a HTTPS protokolů. [4]

2.1.3 Tvorba Java card aplikace

Existuje mnoho vývojových prostředí, lze použít jak klasická prostředí pro vývoj java aplikací (NetBeans, Eclipse) a do nich doinstalovat potřebné pluginy pro Java card vývoj nebo lze použít speciální vývojová prostředí, např. JCIDE.

Běžná java aplikace se kompiluje do souboru .class. Pro Java karty je nutné tento soubor společně s AID identifikátor převést na soubor .cap. Převod umožňují některá vývojová prostředí, případně aplikace ant-javacard.

Zbývá nahrát .cap soubor na kartu. Umožňují to programy GPShell nebo GlobalPlatfromPro (dále GP). Druhý jmenovaný má sice méně funkcí, ale je podstatně

jednodušší [5]. Základní příkazy programu GP:

- GP -info – informace o kartě,
- GP -list – nainstalované applety na kartě, včetně jejich AID,
- GP -delete – smaže applet na kartě,
- GP -install – nainstaluje applet na kartu.

2.1.4 Bezpečnost Java karet

Java karty byly vyvinuty s ohledem na bezpečnost dat na nich uložených. Bezpečnost je zajištěna pomocí následujících mechanizmů:

- zapouzdření dat – data jsou uložena uvnitř aplikace a spouštěna v izolovaném prostředí (JCVM), odděleně od operačního systému a hardwaru,
- applet firewall – na kartě může být zároveň více appletů, firewall kontroluje předávání dat mezi applety,
- kryptografie – podpora množství kryptografických primitiv, symetrické šifry, asymetrické šifry, digitální podpisy a výměna klíče,
- applet – applet je stavový stroj, který vykonává pouze příjem příkazu a odesílá odpovědi na dané příkazy.

Jelikož applety běží na velmi limitovaném hardwaru, nelze využívat všechny vlastnosti jazyka Java. Chybí např.:

- dynamické načítání tříd,
- garbage collector (volitelný),
- třída SecurityManager,
- podpora více vláken,
- klonování objektů,
- datové typy string, double, float a long,
- datový typ int (volitelný) [6].

2.2 Basic Card

Jedná se o programovatelné čipové karty obsahující P-code interpreter optimalizovaný pro spouštění programů napsaných v jazyce ZC-Basic. Vývojáři se snažili dosáhnout 3 hlavních cílů:

- Jednoduché k programování – Basic je známý a jednoduchý jazyk, který se dá rychle naučit. Příkaz od terminálu kartě se volá jako klasická Basic funkce. Vývojové prostředí je jednoduché a přehledné. EEPROM data se čtou a zapisují stejně jako RAM data.
- Bezpečnost – karty podporují velké množství kryptografických funkcí. Z asymetrických šifer je to RSA a EC, symetrické DES a AES, hashovací funkce SHA-1 až SHA-512. Šifrování se zaručením autentičnosti EAX a digitální podpis OMAC.
- ISO kompatibilní – na Basic kartách je možné definovat vlastní ISO kompatibilní příkazy stejně jako klasickou funkci.

Basic karty podporují komunikační protokoly T=0, T=1 a bezkontaktní T=CL. Operační systém obsahuje předdefinované příkazy pro nahrávání aplikací na kartu, případně umožňující zapnout automatické šifrování APDU komunikace.

Zdrojový kód je kompilátorem ZCMBasic převeden na P-code, což je jazyk, který je interpretován virtuálním strojem. P-code se nahrává na kartu nástrojem BCLoad Card Loader nebo ZCMDCard debbuger (děje se na pozadí ve vývojovém prostředí).

Basic karty (od verze ZC7 a ZC8 rev. D) mohou být nakonfigurovány, aby se chovaly jako Mifare karty. Karta se poté rozhodne dle typu čtečky, jak bude fungovat. Pokud je ve čtečce pro Basic karty, lze k Mifare datům přistupovat stejně jako k ostatním datům na kartě [7].

2.2.1 Basic Card aplikace a soubory

Basic Card aplikace je uložena v jednom zdrojovém souboru, který ale může načítat další soubory. Soubor má příponu .BAS a skládá se ze sady příkazů pracujících s proměnnými a soubory. Všechny příkazy mohou číst a zapisovat do souborů a proměnných asociovaných dané aplikaci. Karty z řad *Enhanced* a *Professional* mohou obsahovat pouze jednu aplikaci. Karty z řady *MultiApplication* mohou obsahovat až 128 aplikací, jednotlivé aplikace mají přístup pouze k vlastním datům, ale k souborům může přistupovat více různých aplikací dle nastavených práv.

Kompilátor dokáže ze zdrojového .BAS souboru udělat .IMG soubor. Tento soubor je možné nahrát na kartu nebo spustit v P-code interpreteru ZCMSim.

V případě, že je aplikace spuštěna v debugeru ZCMDCard, kompilátor vytvoří ještě debugovací soubor .DBG, což je obraz (.IMG) včetně debugovacích informací. Program ZCMDCard pracuje se simulovanými Basic kartami. Simulované karty jsou uložené v souborech .ZCC, které obsahují simulovanou EEPROM paměť.

2.2.2 Struktura aplikací

Basic Card aplikace se skládá ze tří částí [7].

- Inicializační kód – první blok kódu, který není součástí žádné procedury. U karet s jednou aplikací se spouští v čase zavolání prvního příkazu z terminálu a multiaplikačních karet se spouští v momentě vybrání aplikace. Není povinný a používá se např. pro kontrolu, jestli karta není zablokována vydavatelem nebo jestli jsou na kartě soubory potřebné pro funkci aplikace.
- Definice procedur – jazyk ZC-Basic má tři typy procedur – podprogram, funkce a příkaz. Podprogramy a funkce jsou stejné jako v jazyce Basic. Podprogram je blok kódu spustitelný z jiných procedur. Funkce je stejná jako podprogram, ale umožňuje vracet hodnotu. Příkaz (jak je chápán v ZC-Basic) je způsob, jakým komunikuje terminál s aplikací na kartě.

  Mezi klíčovým slovem příkaz (command) a názvem příkazu se nachází dva bajty nazývané CLA a INS, dle kterých karta určí, jaký příkaz má vykonat.

- Definice souborů – soubory mohou být vytvářeny jak programovým kódem během vykonávání aplikace na kartě, tak v terminálu. Speciálním případem je vytváření souborů v tzv. sekci souborových definic. Tato sekce začíná

klíčovým slovem *Dir* a končí *End dir*. Soubory v ní definované se vytvoří při kompilaci a poté jsou nahrány na kartu spolu s aplikací.

2.3 .NET smart card

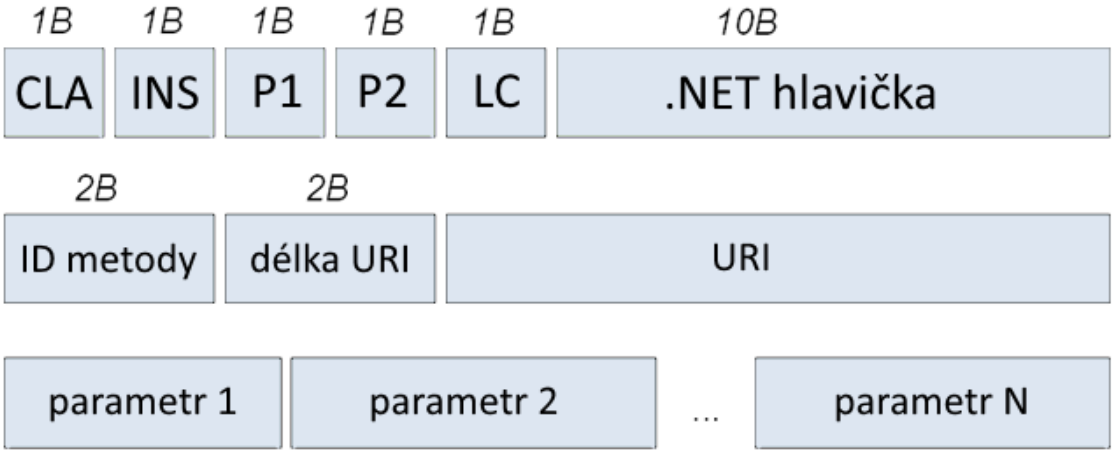
Pro tyto karty lze programovat aplikace v jazycích kompatibilních s .NET frameworkem – C# nebo VB.NET. Vývojový kit je součástí Visual Studia. Aplikace se nejprve kompiluje do .NET Intermediate code (CIL) a poté je na kartě převedena do strojového kódu pomocí CLR (Common Language Runtime).

CLR se stará na kartě o mnoho věcí souvisejících s během aplikací:

- oddělení běžících aplikací,
- garbage collection – uvolňování paměti,
- bezpečnou komunikaci mezi aplikacemi,
- ošetření výjimek.

Na rozdíl od Java karet, .NET karty z pohledu vývojáře nekomunikují pomocí APDU příkazů. Karta se tváří jako server s vlastním URI (Uniform Resource Identifikator), jehož metody se volají. APDU se používají jako komunikace na nižší vrstvě. O překlad metod na APDU se starají příslušné dll knihovny [8].

Jak se URI dotaz vloží do APDU zprávy je zobrazeno na Obr. 7.



Obr. 7: URI v APDU

- APDU hlavička – bajty CLA, INS, volitelné parametry P1, P2 a bajte LC, který udává délku příkazu.
- .NET hlavička – neměnná pro danou sérii karet, blíže nezdokumentována.
- ID metody – unikátní identifikátor dané metody, náhodně generovaný při kompilaci.
- Délka URI – délka identifikátoru URI.
- URI – délka pole záleží na délce celého URI, (např. 18 bajtů má CryptoService.uri).
- Parametry dané metody – není použit žádný popis datového typu parametru, ani oddělovač jednotlivých parametrů. Datový typ i jeho délka je odvozena

od očekávaného vstupu pro metodu.

- String – 2 bajty délka stringu + string v ASCII kódování bez nulového ukončovacího bajtu.
- Integer – 4 bajty s odpovídající hodnotou.
- Boolean – 2 bajty (např. 01 01 pro hodnotu *true*).

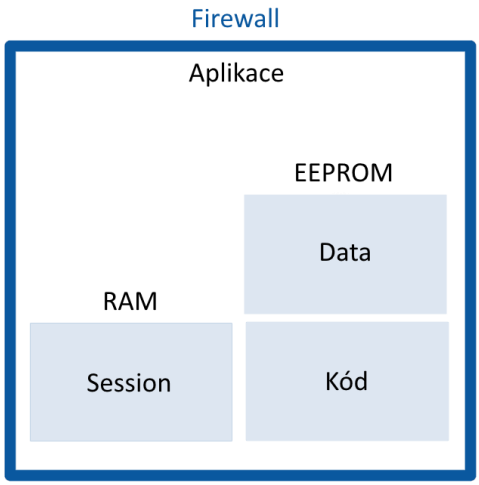
Z bezpečnostního hlediska .NET karty nevynucují podpis ani šifrování APDU příkazů. Takže je možné je odposlechnout, popř. změnit během přenosu.

2.4 MULTOS

Pro operační systém MULTOS lze programovat aplikace v jazyce MEL (MULTOS Executable Language), případně v C nebo Javě a kód je poté zkompilován do MEL [9].

2.4.1 Aplikace MULTOS

MULTOS se snaží, stejně jako ostatní operační systémy, jednotlivé aplikace softwarově oddělovat a sdílet mezi nimi hardwarové prostředky.



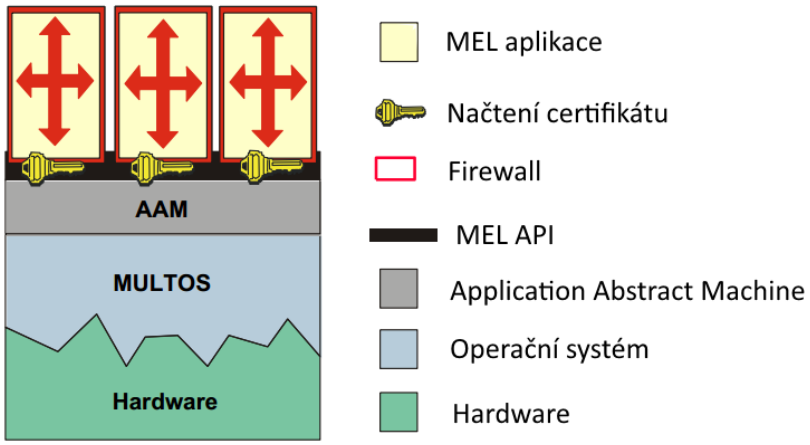
Obr. 8: MULTOS aplikace

- Kód – statická paměť, kde se nachází spustitelný kód aplikace. Do této paměti nelze zapisovat.
- Data – statická paměť s daty, která jsou potřeba pro běh aplikace, z paměti lze číst i do ní zapisovat, ale nikoli z ní spouštět kód.
- Session – dočasná data v operační paměti RAM, např. hodnoty proměnných.

Všechny komponenty (viz Obr. 8) jsou ohraničeny firewallem, který brání jejich neautorizovanému přístupu k datům jiných aplikací a opačně.

2.4.2 Architektura MULTOS

MULTOS se skládá z následujících vrstev (viz Obr. 9):



Obr. 9: Operační systém MULTOS

- Hardware – fyzická platforma podporující funkce operačního systému. Funkce jsou psány ve strojovém kódu a přistupuje k nim virtuální stroj, který není závislý na použitém hardwaru.
- Operační systém – stará se o komunikaci mezi procesy, správu paměti a virtuální stroje. Řídí načítání a mazání aplikací a správu APDU příkazů i odpovědí.
- Application Abstract Machine – poskytuje standardní API skládající se z instrukcí a vestavěných funkcí, primitiv. Virtuální stroj pro běh aplikací.
- MEL API – poskytuje podporu pro MULTOS Assembly Language kód.
- Načtení certifikátů – pro načtení aplikace je potřeba digitálně podepsaný certifikát.
- Aplikace a firewally.

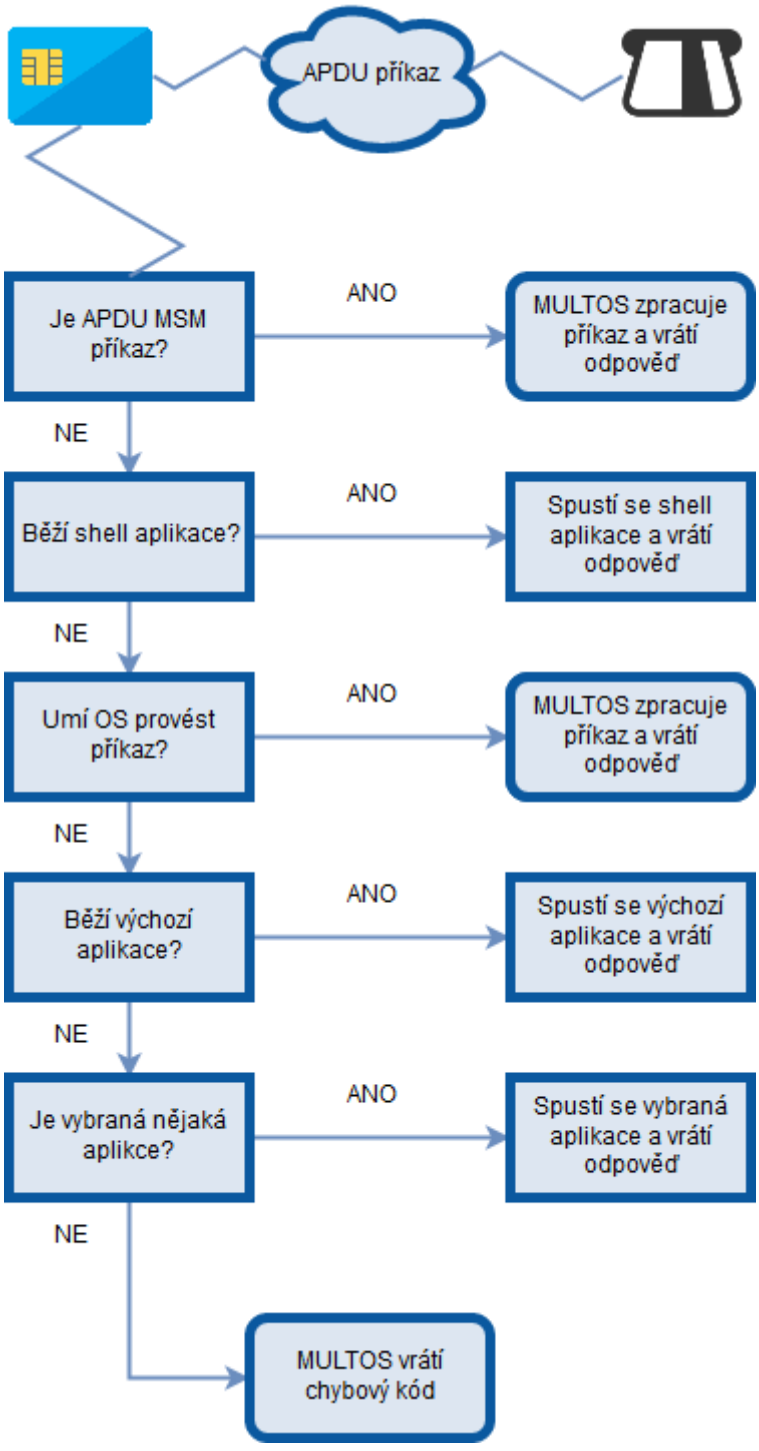
2.4.3 Způsoby provozu

Aplikace v MULTOS systému mohou pracovat v několika módech.

- Standardní mód – běžný mód pro práci s multiaplikační kartou, na kartě jsou různé aplikace a jedna z nich je aktuálně vybrána a může zpracovávat příkazy a odesílat odpovědi.
- Shell – standardní mód předpokládá, že je dostupná čtečka a terminálová aplikace, která dokáže komunikovat s multiaplikační kartou. Jestliže toto není dostupné, všechny příkazy jsou zpracovávány shellem. Shell se tváří jako běžná čipová karta s jednou aplikací, není třeba nejprve vybrat aplikaci dle jejího ID. Shell musí umět sám přesměrovávat příkazy aplikacím, pro které jsou určené.
- Výchozí mód – velmi podobný shell módu, rozdíl je v tom, že terminál komunikuje přímo s výchozí aplikací, ale stále je dostupný příkaz *SELECT FILE* pro zvolení jiné aplikace.

2.4.4 Směrování příkazů

Jednou z důležitých vlastností operačního systému je směrování příkazů. Bez něj by nešlo vybírat aplikace a zasílat jim příkazy.



Obr. 10: Směrování příkazu v operačním systému MULTOS

Vysvětlení k Obr. 10. MSM příkaz v prvním kroku je příkaz, který slouží k načtení nebo smazání aplikace. Krok 3 – kontrola, jestli příkaz není určen operačnímu systému – obvykle příkaz pro volbu aplikace *SELECT FILE*. [10]

2.5 Kryptografické funkce poskytované operačními systémy čipových karet

Všechny operační systémy na čipových kartách se zaměřují na poskytování co nejvíce kryptografických funkcí, jejich optimalizaci a dostatečnou bezpečnost zajištěnou správnou implementací a délkou klíčů. Čipové karty totiž ve většině případů pracují s citlivými osobními údaji a není žádoucí jejich kompromitace nebo odposlech. Srovnání v Tab. 2.

Tab. 2: Srovnání kryptografických funkcí na OS

		Basic	.NET	Java	MultOS
Symterické šifry	DES, 3DES	ano	ano	ano	ano
	AES	ano (až 256b)	ano (až 256b)	ano (až 256b)	ano
Asymetrické šifry	RSA	ano (až 4096b)	ano (až 2048b)	ano (až 4096b)	ano (až 2048b)
	EC	ano (až 544b)		ano (až 512b)	ano (až 512b)
Hashovací funkce	SHA1	ano	ano	ano	ano
	SHA2	ano (až 512b)	ano (až 256b)	ano (až 512b)	ano (až 256b)
	MD5		ano	ano	
Digitální podpis	MAC	ano (OMAC)	ano (HMACSHA1)	ano (HMAC)	pouze RSA
	DSA	ano (ECDSA)		ano (i ECDSA)	
Ustanovení klíče	ECDH	ano		ano	
Modulární aritmetika		ano (od verze 7 rev. C)	ne	ne	ano
Generace náhodných čísel		PRNG, od verze 7 TRNG	TRNG	TRNG	dle implementace výrobce
Ostatní		EAX, ECNR		RIPEMD-160	

Tabulka vychází z dat udávaných tvůrci operačních systémů v dokumentech pro vývojáře a prezentačních materiálech. Ne vždy karty vyrábí a prodává tvůrce operačního systému a u jiných výrobců není záruka, že operační systém bude umět všechny kryptografické funkce. Důkazem je asi nejrozšířenější OS Java card, kde různí výrobci čipových karet (i když podporují stejnou verzi Java API) podporují různé kryptografické funkce. Přičemž ve většině případů výrobci karet funkce omezují, než že by přidávali vlastní.

Hlavními zdroji dat jsou:

- Basic card – developers guide v8.15 [7]
- .NET card – integration guide D1265202C [11]
- Java card – JAPI 3.0.5 [12]
- MultOS card – developers guide v1.41 [10]

3 PODROBNÉ SROVNÁNÍ JAVA CARD A BASIC CARD

Java karty patří mezi velmi rozšířený druh karet a Basic karty zase umožňují jednoduché programování a nabízejí širokou podporu kryptografických funkcí, proto jsem se rozhodl podrobněji srovnat tyto dvě platformy.

3.1 Rychlost přenosu dat a zpracování APDU

Jelikož se budu v práci dále zabývat rychlostí provádění kryptografických funkcí na daných kartách, rozhodl jsem se změřit i rychlost zpracování prázdných APDU a APDU s různým množstvím dat, jak při posílání dat na kartu, tak při čtení dat z karty.

Pro měření jsem měl k dispozici kontaktní Java kartu NXP J3D081, kontaktní Basic kartu ZC7.6 rev. D a bezkontaktní Basic kartu ZC7.5 rev. A.

Měřil jsem reakci karty na zaslání prázdného APDU a poté reakce na APDU s daty (8, 16, 32, 64, 128 a 256 bajtů). Při čtení dat z karty jsem postupoval obdobně. Výsledné hodnoty jsou vždy průměrem z 10 měření. Výsledky jsou znázorněny v Tab. 3 pro posílání dat a v Tab. 4 pro čtení dat.

Tab. 3: Posílání APDU s daty na kartu

	0B [ms]	8B [ms]	16B [ms]	32B [ms]	64B [ms]	128B [ms]	256B [ms]
Basic card ZC7.5 rev.A - CL	3,3	5	5,1	6,6	9	13,1	23,2
Basic card ZC7.6 rev.D	4,2	5,2	7	9,4	14,4	25	49,8
Java card NXP J3D081	11,9	14	15,3	18,4	25,4	40,4	72,6

Tab. 4: Čtení dat z karty

	0B [ms]	8B [ms]	16B [ms]	32B [ms]	64B [ms]	128B [ms]	256B [ms]
Basic card ZC7.5 rev.A - CL	4,4	4,9	6,1	7,2	9,5	14	21,3
Basic card ZC7.6 rev.D	4,4	5,2	5,8	6,4	9,8	14,6	26,4
Java card NXP J3D081	12,8	13,1	15,8	18,9	27,1	40,5	73

Rychlost přenosu dat spočteme z APDU s největším množstvím dat jako:

$$R = \frac{256}{A_{256} - A_0} * 1000 * 8\ [bps],$$

kde A_{256} je doba přenosu APDU s daty o velikosti 256 bajtů, A_0 je doba přenosu APDU bez dat. Konstanta 256 je počet datových bajtů A_{256}, konstanta 1000 je použita pro převod milisekund na sekundy a konstanta 8 je pro převod z bajtů na bity.

U bezkontaktní Basic card ZC7.5 rev.A – CL získáváme rychlost přenosu dat při posílání z terminálu na kartu:

$$R = \frac{256}{23{,}2 - 3{,}3} * 1000 * 8 \cong 102915\ bps.$$

Tato hodnota odpovídá standardu pro bezkontaktní přenos dat ISO/IEC 14443, kde je minimální rychlost přenosu 106 kbps.

S kontaktní Basic card ZC7.6 rev. D bylo dosaženo rychlosti 44912 bps při posílání dat z terminálu a rychlosti 93090 bps při čtení dat z karty. Výrobce udává u těchto karet rychlost 38400 baudů. Naměřená rychlost tedy může ukazovat na vícestavovou modulaci, přičemž je pravděpodobně použita jiná modulace při zápisu a jiná při čtení dat.

Obdobný výpočet provedeme pro Java card NXP J3D081:

$$R = \frac{256}{72{,}6 - 11{,}9} * 1000 * 8 \cong 33739\ bps.$$

Rychlost na Java kartě odpovídá rychlosti udávané výrobcem – 38400 bps.

Z uvedených rychlostí plyne zjištění, že bezkontaktní karty jsou při přenosu v ideálním prostředí několikanásobně rychlejší, než karty kontaktní. Dále je z Tab. 3 a Tab. 4 patrné, že rychlosti čtení a zápisu jsou u Basic card ZC7.5 rev. A a Java card NXP J3D081 téměř identické.

3.2 Rychlost výpočtu kryptografických funkcí

Rychlost provádění kryptografických operací na čipových kartách je velmi důležitá. Lidé při použití karet (především bezkontaktních) nechtějí a nejsou zvyklí čekat na odezvu terminálu. Z tohoto důvodu je potřeba, aby se všechny potřebné operace pro bezpečnou komunikaci provedly v krátkém čase, ideálně pod 1 sekundu.

Měření bylo prováděno na kartě Basic card ZC7.6 rev. D, která již podporuje knihovnu Crypto.DEF a je možné na ní měřit velké množství kryptografických primitiv. Na kartě Java card NXP J3D081 bylo měření prováděno pomocí aplikace JcAlgTest.

Naměřené výsledky jsou v Tab. 5 a Tab. 6.

Tab. 5: Měření rychlosti kryptografických funkcí - část 1

	SHA1 [ms]	SHA256 [ms]	DES [ms]	3DES [ms]	AES128 [ms]	AES256 [ms]	Random number [ms]
Basic card ZC7.6 rev.D	42	61	1,6	1,6	1,6	1,6	13
Java card NXP J3D081	11,7	21,2	6	7,1	6,7	7,7	10,4

Tab. 6: Měření rychlosti kryptografických funkcí - část 2

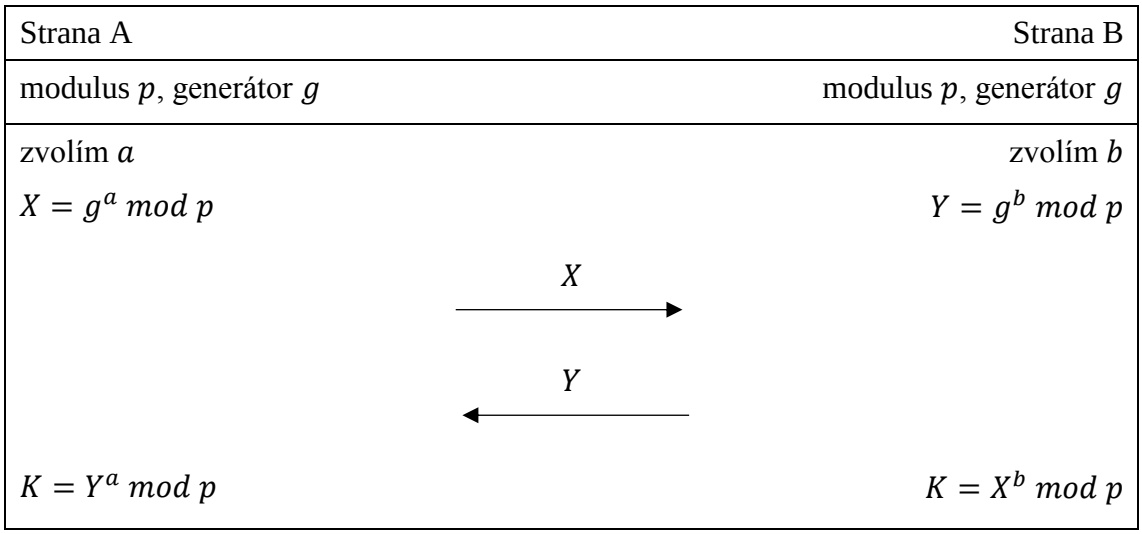
	RSA gen 1024 [ms]	RSA enc 1024 [ms]	RSA dec 1024 [ms]	RSA gen 2048 [ms]	RSA enc 2048 [ms]	RSA dec 2048 [ms]
Basic card ZC7.6 rev.D	2241	25	72	13079	33	397
Java card NXP J3D081	3387	8,9	116	23511	17,7	595

Z měření je patrné, že karty dosahují řádově podobných výsledků. Nelze jednoznačně určit, která z karet dosahuje vyšších rychlostí. Basic karta je rychlejší při provádění symetrických šifer DES, 3DES a AES, a také při generaci klíče a dešifrování RSA. Naopak Java karta je rychlejší při hashování SHA1 a SHA256, při generování bezpečných náhodných čísel a při operaci šifrování RSA.

Rozdíly v rychlosti jsou způsobené především rozdílnou softwarovou implementací daných algoritmů a lišit se mohou i hardwarové obvody na kartě určené pro akceleraci kryptografických výpočtů.

4 PAKE (PASSWORD AUTHENTICATED KEY EXCHANGE)

Jedná se o ustanovení klíče na základě předsdíleného hesla. Pokud mluvíme o ustanovení klíče mezi dvěma stranami, obvykle spolu tyto strany komunikují přes nezabezpečený kanál, a proto nepřipadá v úvahu, že by si šifrovací klíč jednoduše poslaly. Nejpoužívanější algoritmus pro ustanovení klíče přes nezabezpečený kanál je Diffie-Hellman (DH) - Obr. 11.



Obr. 11: Postup ustanovení klíče DH

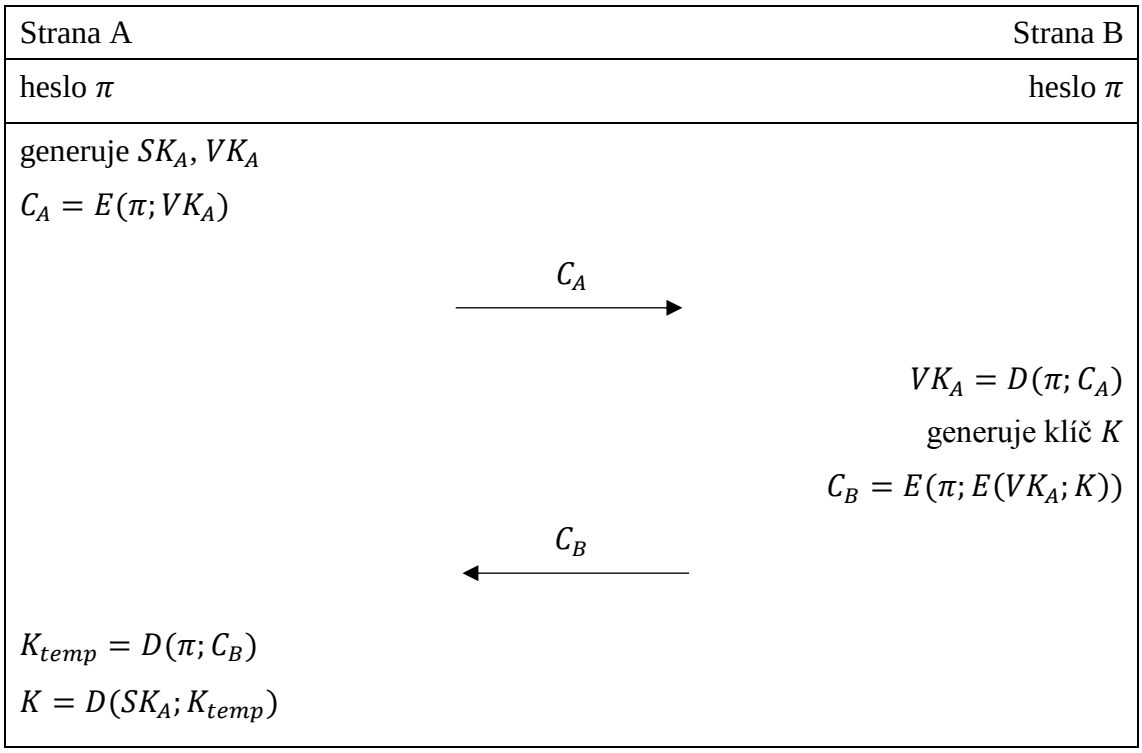
Pokud je útočník pouze pasivní a odposlouchává komunikaci, není schopný zjistit heslo K, musel by dokázat efektivně řešit problém diskrétního logaritmu. Ovšem aktivní útočník by mohl navázat komunikaci zvlášť se stranou A i B a následně komunikaci odposlouchávat. U DH protokolu totiž není řešena autentizace stran. Řešením tohoto problému jsou právě algoritmy PAKE, které využívají předsdílené heslo. Pokud jedna strana dokáže druhé, že vlastní správné heslo, předpokládáme, že nekomunikujeme s útočníkem. Zároveň toto předsdílené heslo nelze použít přímo pro šifrování komunikace, protože často musí být snadno zapamatovatelné a tudíž uživatelé volí krátká hesla a hesla, která lze snadno prolomit pomocí slovníkového útoku.

4.1 EKE (Encrypted Key Exchange)

Jeden z prvních protokolů PAKE je právě Encrypted Key Exchange. Byl publikován a patentován v roce 1992. Roku 2013 patent vypršel.

Obvyklá výměna klíče s pomocí předsdíleného hesla (π) probíhá tak, že strana A vygeneruje náhodný klíč, zašifruje jej pomocí π a odešle, strana B si vygenerovaný klíče dešifruje a může ho použít. Problém je v tom, že pokud útočník komunikaci odchytne, může slovníkovým útokem nebo útokem hrubou silou prolomit heslo π a tím pádem získat klíč, kterým je šifrovaná probíhající komunikace.

Protokol EKE funguje tak, že strana A vygeneruje pár klíčů SK_A, VK_A. Kde SK je soukromý klíč a VK je veřejný klíč. Pomocí hesla π zašifruje svůj veřejný klíč a odešle ho straně B. Strana B dešifruje VK_A, vygeneruje klíč K, který zašifruje pomocí VK_A a výsledný kryptogram zašifruje pomocí π a odešle straně A. Strana A provede dvojnásobné dešifrování a získá klíč K (viz Obr. 12). [13]



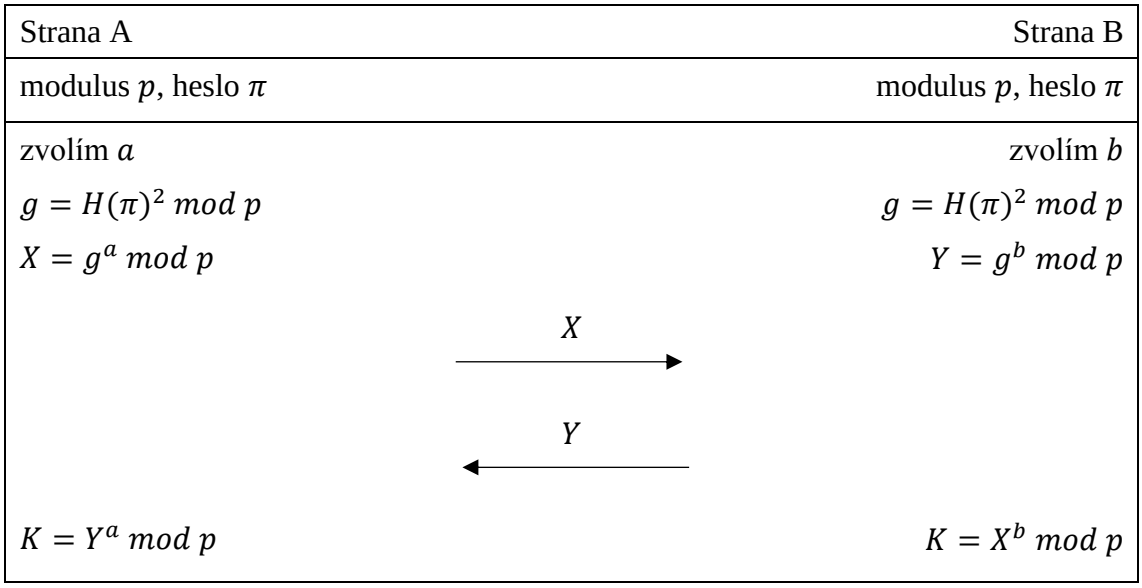
Obr. 12: Ustanovení klíče EKE

I kdyby útočník komunikaci odchytával a povedlo se mu zjistit heslo π, dokáže s jeho pomocí pouze dešifrovat VK_A, ale klíč, který je pomocí VK_A zašifrovaný, už se mu bez znalosti SK_A získat nepodaří. Ani aktivní útok, kdy by útočník chtěl navázat spojení s oběma stranami, nebude fungovat, protože útočník nezná π.

4.2 SPEKE (Simple Password Exponential Key Exchange)

SPEKE patří k nejznámějším protokolům pro ustanovení klíče na základě předsdíleného hesla. Byl představen v roce 1996 a je součástí standardů IEEE P1363.2 and ISO/IEC 11770-4. [14]

Tato metoda je založena na DH protokolu, rozdíl je v tom, že generátor g je odvozen z hashe hesla π.



Obr. 13: Protokol SPEKE

Od roku 1996 bylo publikováno několik útoků na SPEKE [15], v roce 2004 bylo zjištěno, že některá hesla jsou exponenciálně ekvivalentní a tudíž při útoku hrubou silou jde zkoušet víc hesel během jednoho pokusu [16]. Původní výpočet generátoru byl následující:

$$g = \pi^2 \ mod \ p.$$

Po přidání hashovací funkce do výpočtu generátoru je tato zranitelnost eliminována (viz Obr. 13).

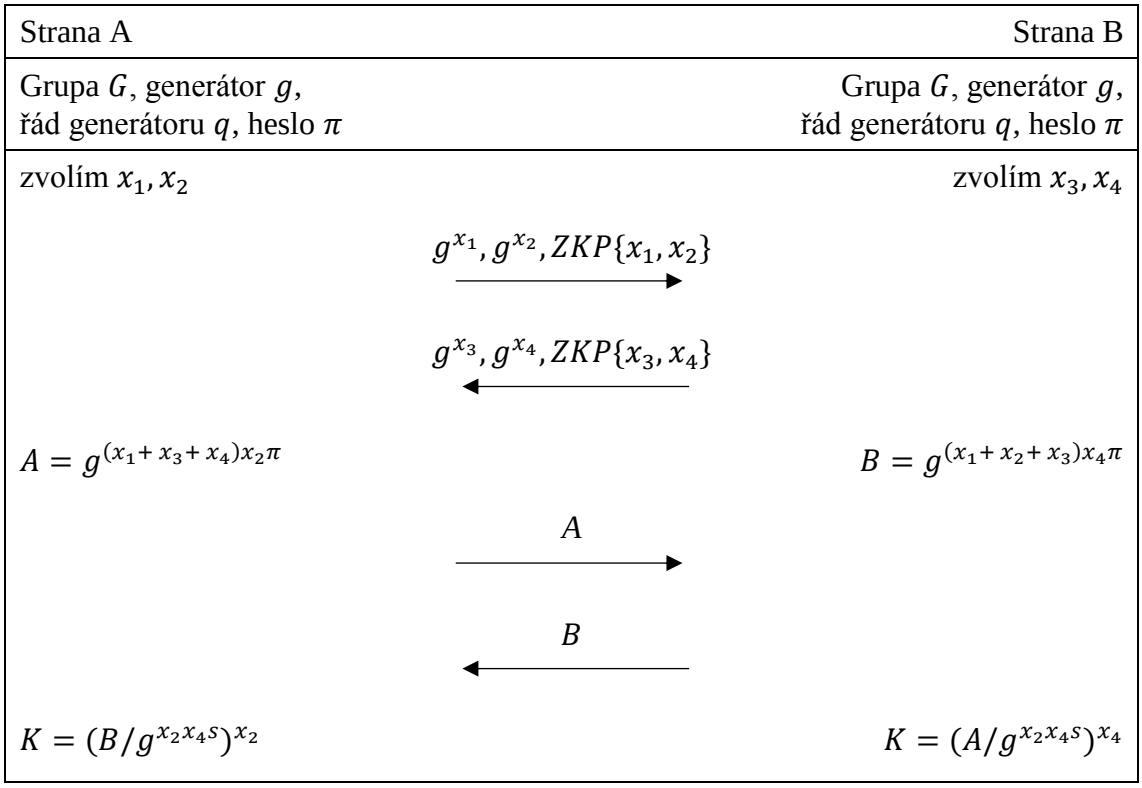
4.3 J-PAKE (Password Authenticated Key Exchange by Juggling)

Protokol představený v roce 2008 [17]. Na rozdíl od předchozích PAKE protokolů J-PAKE využívá protokol ZKP (Zero-Knowledge Proof), což je kryptografický protokol, kterým jedna strana (dokazovatel) dokazuje druhé (ověřovatel) určitou znalost (např. znalost klíče), aniž by tuto informaci musela odtajnit (poslat nezabezpečeným kanálem) [18].

Jako příklad ZKP protokolu může posloužit diskrétní logaritmus. Strana A musí dokázat znalost tajného x, takže vypočítá:

$$y = g^x \ mod \ p$$

a odešle výsledek straně B, která provede stejný výpočet a pokud výsledky souhlasí, je to důkaz znalosti x (viz Obr. 14).



Obr. 14: Protokol J-PAKE

Jako ZKP je použit Schnorrův podpis [19].

4.4 PDM (Password Derived Moduli)

Další PAKE protokol založený na ustanovení klíče DH. Byl vyvinut jako otevřený protokol, protože v době jeho vzniku byl např. hojně využívaný SPEKE chráněný patenty. PDM je velmi podobný SPEKE protokolu, zde ovšem není vytvářen generátor g, ale z uživatelského hesla je deterministickým způsobem spočítáno prvočíslo p. Uživatelské heslo je použito jako semeno generátoru, který generuje náhodná čísla a ty jsou následně testována na prvočíselnost.

Poté, co strana A vypočítá prvočíslo p, zvolí generátor $g = 2$ a dále protokol pokračuje stejně jako DH. Na straně serveru je prvočíslo už předpočítáno, což snižuje nároky na server. [20]

4.5 Implementace PAKE protokolů na čipových kartách

Mezi nejrozšířenější programovatelné čipové karty patří bezpochyby karty platformy Javacard, proto většina implementací PAKE protokolů byla provedena právě na těchto kartách.

Nejnovější verze čipových karet všech známých platforem (Javacard, Basic card, MultOS) mají vedle hlavního procesoru i kryptoprocesor pro urychlení výpočtů potřebných pro kryptografické metody. Bohužel ne vždy může vývojář k tomuto

pomocnému procesoru přistupovat přímo, a pokud nejsou dostupné vyhovující funkce, musí být výpočty prováděny SW cestou, což je velmi zpomaluje.

Příklady implementace PAKE protokolů na čipových kartách.

- Implementace protokolu PDM na Javacard GXP PRO-R3 [21]. Bohužel se ukázalo, že karta není dostatečně výkonná na výpočty modulárního mocnění v uspokojivém čase (až stovky minut), a proto musel být protokol velmi zjednodušen, čímž byla oslabena jeho bezpečnost.
- Implementace protokolů EKE a SPEKE na Javacard od Cyberflex Access [22]. Zde bylo dosaženo výrazně lepších časů, než v předchozím případě, provedení protokolu EKE trvalo přibližně 1.5 s a o něco složitější SPEKE 3.3 s.
- Implementace protokolu SRP-6a (Secure Remote Password Protocol) s 2048b klíčem na kartě JCOP 2.4 [23]. Opět byla implementace úspěšná a medián dosažených časů byl 1.6 s. Stejný tým následně testoval i protokol EC-SRP s Koblitzovou eliptickou křivkou *secp192k1* a podařilo se jim snížit čas ustanovení klíče na 528 ms.

5 PACE (PASSWORD AUTHENTICATED CONNECTION ESTABLISHMENT)

Jedním z PAKE protokolů je i protokol PACE využívající ECDH protokol pro ustanovení klíče. PACE je protokol určený k autentizovanému ustanovení klíče mezi dvěma stranami (v našem případě terminál a čipová karta) po nezabezpečeném kanále. Protokol byl navržen Spolkovým úřadem pro bezpečnost informační techniky (BSI) především pro elektronické občanské průkazy [24].

Autentizované ustanovení klíče je základní kryptografická úloha, kde se dvě komunikující strany potřebují dohodnout na bezpečném klíči pro šifrování komunikace. Běžné ustanovení klíče (např. pomocí DH protokolu) nezaručuje autentizaci dat, což umožňuje útočníkovi dohodnout si klíč s oběma stranami (které si myslí, že komunikují pouze mezi sebou) a následnou komunikaci odposlouchávat.

Hlavní výhodou protokolu PACE je ustanovení bezpečných klíčů nezávisle na bezpečnosti výchozího hesla. Dle BSI stačí jednoduché heslo o délce šesti znaků.

Protokol můžeme rozdělit na 4 hlavní části, kde se využívá mnoho rozličných kryptografických funkcí: hashování, symetrická kryptografie, Diffie-Hellmanovo ustanovení klíčů s využitím eliptických křivek a MAC autentizace (viz Obr. 15).

strana A	strana B
slabé heslo π	slabé heslo π
parametry eliptické křivky $\mathcal{G} = (a, b, p, G, n, h)$	
$K_\pi = \mathcal{H}(0\|\|\pi)$ zvolím náhodné s $z = C(K_\pi, s)$	$K_\pi = \mathcal{H}(0\|\|\pi)$
$\mathcal{G}$, z →	
	$s = C^{-1}(K_\pi, z)$
funkce Map2Point (DH2Point)	
zvolím x_A $X_A = x_A \cdot G$	zvolím x_B $X_B = x_B \cdot G$
X_A →	
← X_B	
$H = x_A \cdot X_B$ $\hat{\mathrm{G}} = sG + H$	$H = x_B \cdot X_A$ $\hat{\mathrm{G}} = sG + H$
zvolím y_A $Y_A = y_A \cdot \hat{\mathrm{G}}$	zvolím y_B $Y_B = y_B \cdot \hat{\mathrm{G}}$
Y_A →	
← Y_B	
$K = y_A \cdot Y_B$	$K = y_B \cdot Y_A$
$K_{enc} = \mathcal{H}(1\|\|K)$ $K_{mac} = \mathcal{H}(2\|\|K)$ $K'_{mac} = \mathcal{H}(3\|\|K)$ $T_A = MAC(K'_{mac}, (Y_B, \mathcal{G}))$	$K_{enc} = \mathcal{H}(1\|\|K)$ $K_{mac} = \mathcal{H}(2\|\|K)$ $K'_{mac} = \mathcal{H}(3\|\|K)$ $T_B = MAC(K'_{mac}, (Y_A, \mathcal{G}))$
klíč - (K_{enc}, K_{mac})	klíč - (K_{enc}, K_{mac})

Obr. 15: Protokol PACE

5.1 První část PACE

Na začátku máme předsdílené heslo π, které je uložené v terminálu i na kartě, stejně jako parametry eliptické křivky $\mathcal{G}$. Eliptická křivka je určena koeficienty a a b, modulem p, generátorem G, řádem generátoru n a kofaktorem h.

Strana A spočte hash hesla π, čímž dostane K_π. Poté vygeneruje náhodné číslo s, které zašifruje pomocí symetrické šifry (např. AES) s využitím klíče K_π a získá kryptogram z. Strana B taky spočte K_π a pomocí něj dešifruje kryptogram z a zjistí číslo s.

5.2 Druhá část PACE

Druhá část protokolu PACE se nazývá Map2Point. Lze ji provést několika způsoby, např. Hash2Point, Coin2Point nebo Power2Point. My ovšem využijeme metodu DH2Point, která je založena na algoritmu ECDH. Diffie-Hellmanovo ustanovení klíče s využitím eliptických křivek.

Kryptografie eliptických křivek je založena na problému diskrétního logaritmu eliptických křivek. Mějme dva body P a Q, na dané eliptické křivce, pro které platí:

$$P = k \cdot Q,$$

kde k je dostatečně velké celé číslo. Pak ze znalosti P a Q nelze v přijatelném čase nalézt číslo k.

V našem konkrétním případě si strana A zvolí náhodné x_A a spočte X_A, které pošle straně B. Strana B si zvolí náhodné x_B a spočte X_B. Spočtené výsledky si strany vymění a získají H. Před výpočtem hodnoty H je potřeba zkontrolovat, jestli se X_A nebo X_B nerovná nule, čímž by se narušila bezpečnost protokolu a musel by být ukončen. Dále využijeme násobení a sčítání bodů na eliptické křivce a získáme výsledek $\hat{\mathrm{G}}$.

5.3 Třetí část PACE

Nyní se opět spočte ECDH algoritmus, ovšem jako generátor eliptické křivky se použije $\hat{\mathrm{G}}$ získané v předchozím kroku. Opět si obě strany zvolí náhodné y_A a y_B a spočtou:

$$Y_A = y_A \cdot \hat{\mathrm{G}} \text{ a } Y_B = y_B \cdot \hat{\mathrm{G}}.$$

Výsledky si vymění a mohou spočítat klíč K:

$$K = y_A \cdot Y_B = y_B \cdot Y_A.$$

5.4 Čtvrtá část PACE

Finální klíč pro šifrování komunikace, K_{enc}, získáme jako hash klíče K zvětšeného o 1. Klíč pro digitální podpis MAC, K_{mac}, získáme jako hash klíče K zvětšeného o 2. Dále vypočítáme K'_{mac} pomocí kterého digitálně podepíšeme $Y_B, \mathcal{G}$ (resp. $Y_A, \mathcal{G}$) a pošleme protistraně jako výsledné ověření správnosti průběhu PACE [25].

6 NÁVRH IMPLEMENTACE PACE NA BASIC CARD

Protokol PACE můžeme realizovat na kartách Basic card. Tyto karty ve verzích 7 a 8 od revize D mají implementované funkce *point addition* a *point multiplication* na eliptických křivkách, což je důležité pro algoritmus DH2Point, jakožto klíčovou část PACE.

Java karty v nejnovější verzi 3.0.5 nabízí podporu PACE protokolu, ovšem tato verze se zatím velmi obtížně shání. MULTOS karty zase nenabízí tak zdařilé vývojové prostředí a implementace by byla složitější. Z těchto důvodů je využita právě platforma Basic card.

Operace na eliptických křivkách jsou hardwarově akcelerované, a tudíž by celý PACE protokol měl probíhat dostatečně rychle.

Odhad rychlosti PACE protokolu na Basic kartách na základě provedených měření jednotlivých kryptografických primitiv (viz Tab. 7):

Tab. 7: Odhad rychlosti PACE protokolu na Basic card 7.6 rev. D.

	čas [ms]
1. SHA256 hash	61
2. AES šifrování	1,6
3. ECDH	270
4. Point multiplication	156
5. Point addition	16
6. ECDH	270
7. 3xSHA256 hash	183
8. OMAC	10
Součet	967,6

Odhadovaná rychlost je 967,6 milisekund, což je přijatelné pro reálné použití.

7 REALIZACE PROTOKOLU PACE

Protokol PACE je realizován na kartách Basic card, které nabízí všechna potřebná kryptografická primitiva. Terminálová část aplikace je psána v jazyce Java. Pro větší přehlednost kódu je použita zelená barva pro kód na čipové kartě a červená barva pro kód terminálové aplikace.

7.1 Implementace ověřovatele

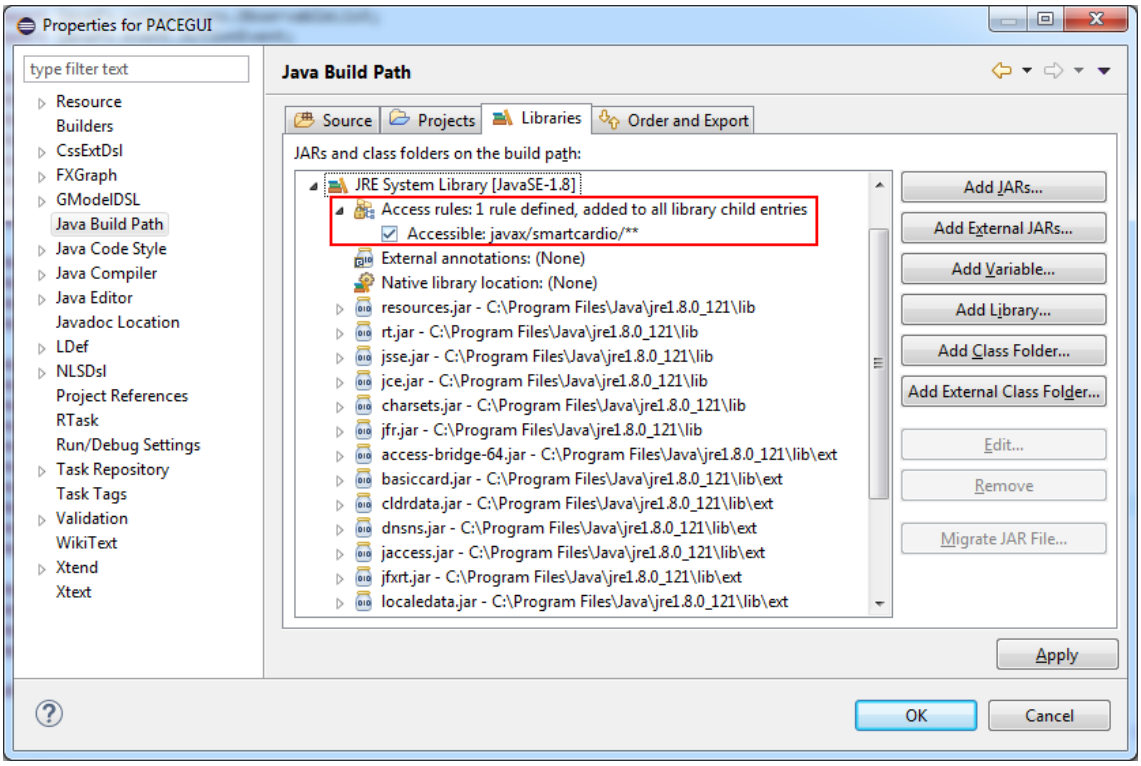
Java nabízí mnohem širší možnosti tvorby grafického uživatelského rozhraní (GUI). Navíc je tento jazyk velmi rozšířený a díky tomu je možné PACE protokol jednoduše integrovat do ostatních aplikací.

Jako vývojové prostředí pro tvorbu Java aplikace byl zvolen software Eclipse. Konkrétně je použita platforma JavaFX a návrh GUI je vytvořen v programu SceneBuilder.

Terminálová aplikace komunikuje s čipovou kartou pomocí příkazů APDU. Výrobce (ZeitControl) sice nabízí i knihovnu pro komunikaci Java aplikací s Basic card, ale poslední verze knihovny je z roku 2005 (v0.95), a proto je zde použita raději přímá komunikace APDU.

7.2 Nastavení vývojového prostředí Eclipse

Pro komunikaci Java aplikace s kartou je použita knihovna Smart Card I/O API. Ve vývojovém prostředí Eclipse musí být tato knihovna povolena v nastavení projektu: Java Build Path – Libraries – JRE Systém Library – Acces rules – Edit – Add – javax/smartcardio/* (viz Obr. 16).



Obr. 16: Povolení knihovny Smart Card I/O

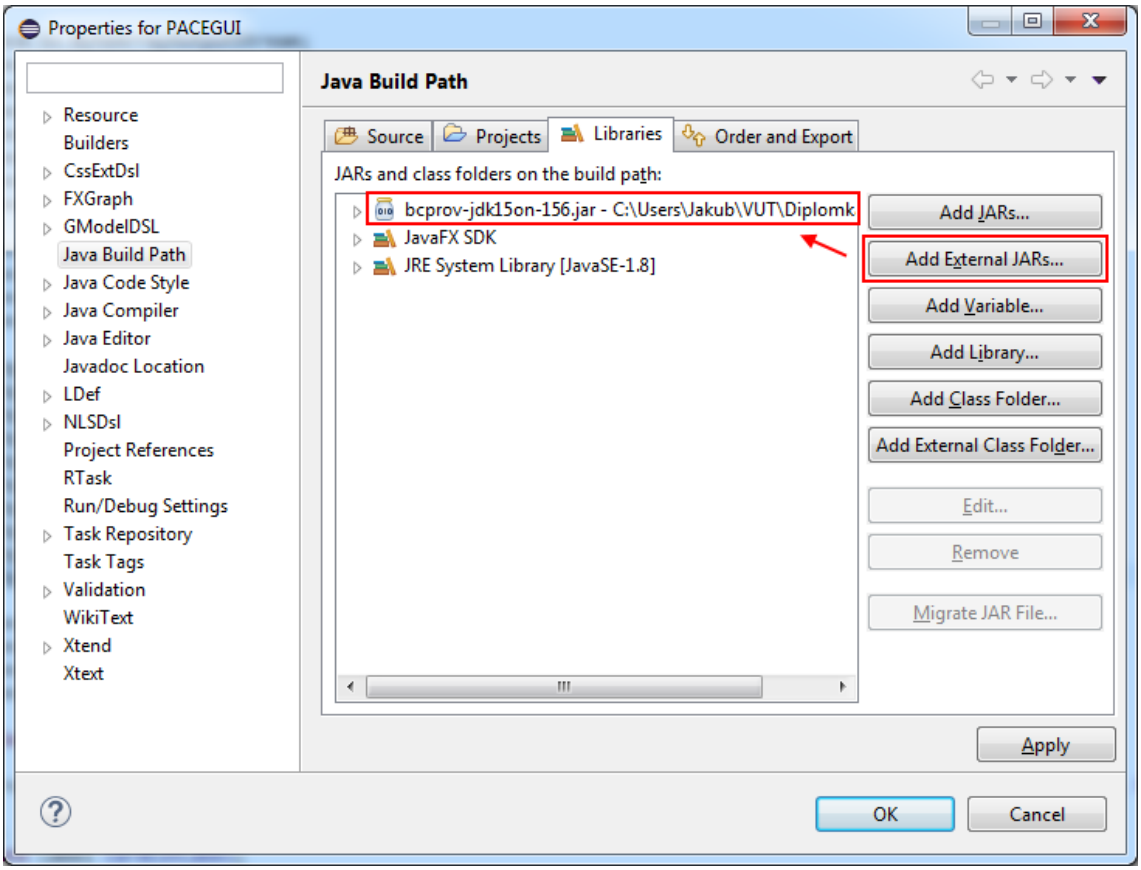
Poté stačí knihovnu importovat.

```
import javax.smartcardio.*;
```

V protokolu PACE bude využito i symetrické šifrování AES s délkou klíče 256b. Základní balík Java Runtime Environment umožňuje použití pouze 128b klíčů, díky politice některých zemí, v kterých je distribuován. Problém lze vyřešit přidáním Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.

Z webu Oracle stáhneme JCE jako zip archiv, ten následně rozbalíme do složky <java-home>/lib/security.

Některé kryptografické operace jsou prováděny pomocí API Bouncy Castle. Jar soubor stažený z webu Bouncy Castle přidáme do Eclipse následujícím způsobem. Nastavení projektu – Java Build Path – Libraries – Add External JARs… (viz Obr. 17).



Obr. 17: Přidání Bouncy Castle API

7.3 Příprava vývojového prostředí BasicCard Development Environment

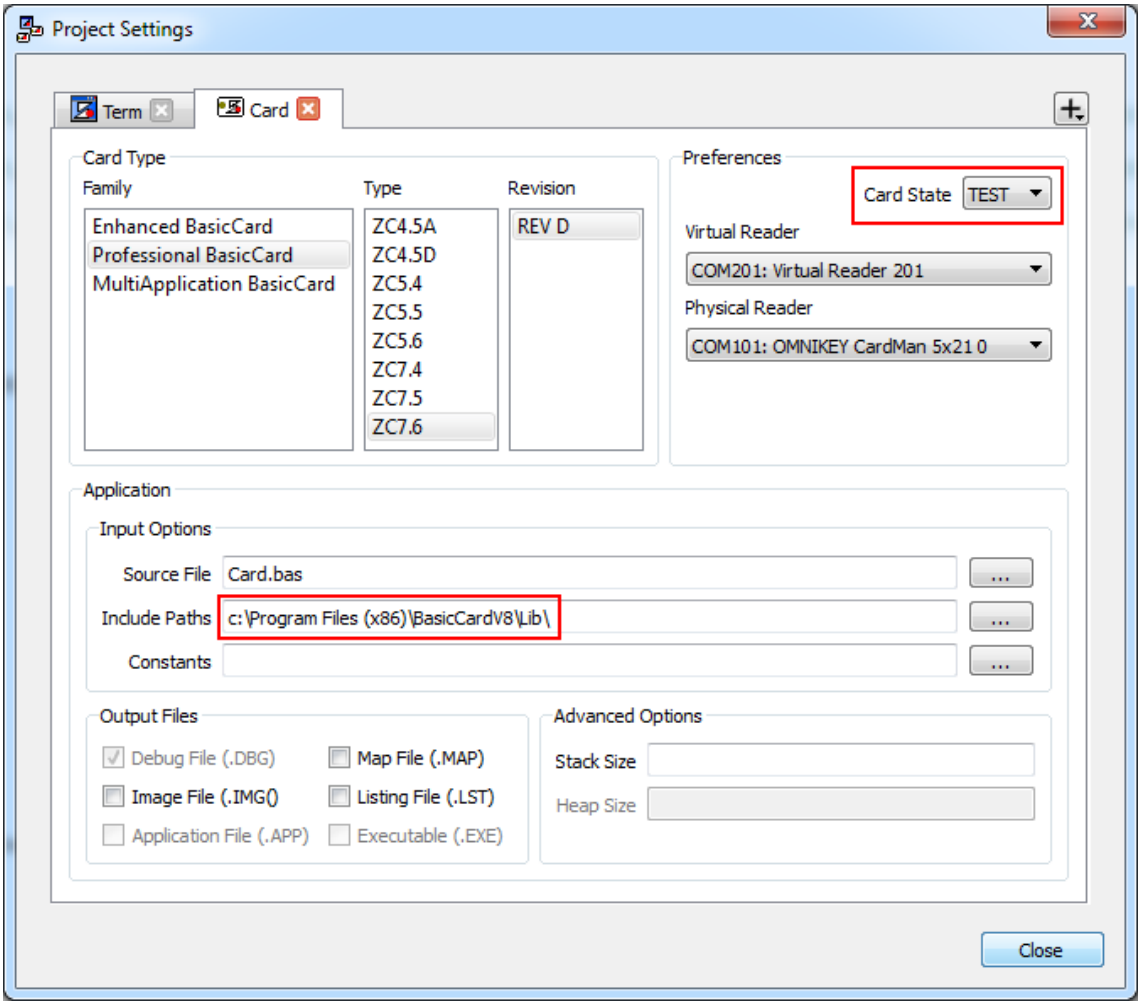
Aplikace pro čipovou kartu byla vyvíjena v prostředí BasicCard Development Environment v8.55. Toto prostředí umožňuje zároveň vývoj, jak aplikace na čipové kartě, tak vývoj terminálové aplikace. Terminálová aplikace byla v tomto prostředí vyvíjena pouze pro testovací účely, výsledná realizace byla prováděna v jazyce Java.

Protokol PACE bude využívat kryptografické funkce definované v souborech *.def, tyto soubory jsou umístěné ve složce <basiccard-home>/lib, obvykle C:\Program Files (x86)\BasicCardV8\Lib\. Proto tuto složku zahrneme do nastavení projektu (viz Obr. 10).

Konkrétně jsou použity následující soubory:

```
#include EC-p.def
#include SHA.DEF
#include AES.DEF
#include OMAC.DEF
```

Při vývoji pro Basic card je důležité mít nastavený „card state“ na hodnotu „test“, aby nedošlo k zablokování karty pro zápis (viz Obr. 18).



Obr. 18: Nastavení Basic card projektu

7.4 Základní komunikace s kartou

V terminálové aplikaci je nejdříve nutné získat seznam dostupných čteček čipových karet.

```
TerminalFactory factory = TerminalFactory.getDefault();
List<CardTerminal> terminals = factory.terminals().list();
```

Následně lze vybrat čtečku, ve které je karta, zvolit transportní protokol a vytvořit spojení.

```
// Volba první čtečky
CardTerminal terminal = terminals.get(0);
// Připojení karty protokolem T=1
Card card = terminal.connect("T=1");
// Vytvoření komunikačního kanálu
CardChannel channel = card.getBasicChannel();
```

Nyní je možné posílat příkazy APDU a číst APDU odpovědi.

```
// APDU příkaz jako string
String apdu = "8801000084";
// Převedení stringu na pole bajtů
Byte [] c1 = DatatypeConverter.parseHexBinary(apdu);
// Odeslání APDU příkazu, odpověď se uloží do proměnné r
ResponseAPDU r = channel.transmit(new CommandAPDU(c1));
```

Rozbor APDU příkazu - 8801000084 :

- CLA = 88,
- INS = 01,
- P1P2 = 0000,
- L_e = 84.

Bajty CLA a INS určují proceduru, která se vykoná na kartě. Bajty P1 a P2 jsou nevyužité. Bajt L_e udává očekávanou délku odpovědi v hexadecimální soustavě. Popsaný APDU příkaz nenese žádná data. Pokud by data nesl, následuje za bajty P1P2 bajt L_c, který udává délku dat, a po něm následují samotná data.

Po dokončení autentizačního protokolu je možné kartu odpojit.

```
card.disconnect(false);
```

7.5 První část PACE

Komunikaci vždy musí inicializovat terminál, v první části se terminálová aplikace dotáže čipové karty na parametry eliptické křivky $\mathcal{G}$ a kryptogram z.

$$K_\pi = \mathcal{H}(0||\pi)$$
$$z = C(K_\pi, s)$$

Nejdříve spočteme SHA-256 hash K_π v terminálu.

```
String heslo = "tajne heslo";
// Inicializace hashovací funkce a tvorba hashe
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(new byte[1]);
md.update(heslo.getBytes());
Byte [] Kpi = md.digest();
```

Poté se dotážeme karty na kryptogram z a rovnou ho dešifrujeme. Náhodné číslo s je 16 bajtové, stejně jako velikost bloku u šifry AES. Očekávaná odpověď APDU bude tedy dlouhá 18 bajtů (16 bajtů kryptogram a 2 bajty SW1 a SW2). $18_{10} = 12_{16} = L_e$. Basic card u funkce AES() využívá mód ECB bez paddingu, tudíž stejné parametry musíme nastavit i v terminálové Java aplikaci. Jelikož je klíč tvořen SHA-256 hashem, má 256 bitů. Aplikace dle délky klíče určí délku šifry, tedy AES 256.

```
String apdu = "8800000012";
Byte [] c1 = DatatypeConverter.parseHexBinary(apdu);
ResponseAPDU r = channel.transmit(new CommandAPDU(c1));

// Tvorba klíče
SecretKeySpec skeySpec = new SecretKeySpec(Kpi, "AES");
// Inicializace AES šifry
Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding");
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
Byte [] decrypteds;
// Dešifrujeme prvních 16b odpovědi
decrypteds = cipher.doFinal(Arrays.copyOfRange(r.getBytes(),0,16));
```

Nyní se můžeme podívat, co dělá příkaz „8800“ na kartě. Nejprve vygenerujeme náhodné číslo s. Basic card na generování náhodných čísel používá funkci *Rnd*, která generuje datový typ long. Funkcí *Hex$()* převedeme typ long na string bajtů a poté funkcí *Mid$()* vezmeme ze stringu prvních 16 bajtů. Nakonec s zašifrujeme AES šifrou (klíčem je opět hash K_π).

```
Command &H88 &H00 Getz(Lc=0, Data$)
  s$ = Hex$(Rnd) + Hex$(Rnd) + Hex$(Rnd)
  s$ = Mid$(s$,1,16)
  Data$ = AES(256,Sha256Hash(Chr$(&H00) + heslo$), s$)
End Command
```

Následně se dotážeme karty na eliptickou křivku. Basic card značí křivky hodnotami 1-19. V ukázkovém kódu je použita křivka NIST P-512, která je na kartě reprezentována číselným označením 19.

```
String apdu = "8801000004";
Byte [] c1 = DatatypeConverter.parseHexBinary(apdu);
ResponseAPDU r = channel.transmit(c1));
// Z odpovědi nám stačí první 2 bajty, které převedeme na string
String temp = Hex.toHexString(Arrays.copyOfRange(r.getBytes(), 0,
2));
// Vytvoření integeru z odpovědi, je nutné převést číslo do
// desítkové soustavy
int i = Integer.parseInt(temp, 16);
// Inicializace příslušně eliptické křivky
ECParameterSpec params = ECNamedCurveTable.getParameterSpec("P-521");
ECCurve curve = params.getCurve();
```

Na kartě je následující funkce.

```
Command &H88 &H01 GetEC(Lc=0, Data%)
  Data% = CurveIndex%
End Command
```

7.6 Druhá část PACE

Jedná se o nejnáročnější část protokolu, na obou stranách dochází 3x k násobení bodu na eliptické křivce konstantou a jednou ke sčítání bodů na křivce.

V terminálu vygenerujeme náhodné číslo x_B, které po vynásobení generátorem eliptické křivky G odešleme na kartu.

$$X_B = x_B \cdot G$$

```
Byte [] xb = new byte[16];
// Vygenerování náhodného pole
new Random().nextBytes(xb);
// Uložení generátoru křivky jako bod G
ECPoint G = params.getG();
// Násobení generátoru náhodným xb
Xb = G.multiply(new BigInteger(xb));
// Souřadnice x a y načteme separátně a v případě potřeby doplníme
// zepředu nulami na 66B (pouze pro P-512) kvůli pozdějšímu
// zpracování na kartě. Na zarovnání použita vlastní funkce
// padding(String s, int l), kde vstupem je string a počet znaků,
// na které má být zleva zarovnán nulami, výstup je zarovnaný string.
String X = padding(Xb.normalize().getXCoord().toString, 132);
String Y = padding(Xb.normalize().getYCoord().toString, 132);
// Odeslání na kartu
String apdu = "8802000084" + X + Y;
Byte [] c1 = DatatypeConverter.parseHexBinary(apdu);
ResponseAPDU r = channel.transmit(c1));
```

Na kartě probíhá pouze uložení bodu X_B do proměnné. Karta už zná hodnotu X_B, nyní zjistíme hodnotu X_A, kterou spočítala karta.

$$X_A = x_A \cdot G$$

Pro výpočet je potřeba znát hodnotu generátoru eliptické křivky, bohužel Basic card neumí k této hodnotě přistupovat, a proto je nutné ji předem uložit do EEPROM paměti karty. Generátor je uložen do proměnné typu string, opět jsou nejprve doplněny obě souřadnice na 66B zepředu nulami a poté v uvedeném pořadí uloženy. Při ukládání je použita funkce Chr$(), jenž vrátí ASCII znak odpovídající udanému kódu. Notace „&H“ udává, že jde o hexadecimální hodnotu.

```
EEPROM G512N$ =
Chr$(&H00,&HC6,&H85,&H8E,&H06,&HB7,&H04,&H04,&HE9,&HCD,&H9E,&H3E,&H
CB,&H66,&H23,&H95,&HB4,&H42,&H9C,&H64,&H81,&H39,&H05,&H3F,&HB5,&H21
,&HF8,&H28,&HAF,&H60,&H6B,&H4D,&H3D,&HBA,&HA1,&H4B,&H5E,&H77,&HEF,&
HE7,&H59,&H28,&HFE,&H1D,&HC1,&H27,&HA2,&HFF,&HA8,&HDE,&H33,&H48,&HB
3,&HC1,&H85,&H6A,&H42,&H9B,&HF9,&H7E,&H7E,&H31,&HC2,&HE5,&HBD,&H66,
&H01,&H18,&H39,&H29,&H6A,&H78,&H9A,&H3B,&HC0,&H04,&H5C,&H8A,&H5F,&H
B4,&H2C,&H7D,&H1B,&HD9,&H98,&HF5,&H44,&H49,&H57,&H9B,&H44,&H68,&H17
,&HAF,&HBD,&H17,&H27,&H3E,&H66,&H2C,&H97,&HEE,&H72,&H99,&H5E,&HF4,&
H26,&H40,&HC5,&H50,&HB9,&H01,&H3F,&HAD,&H07,&H61,&H35,&H3C,&H70,&H8
6,&HA2,&H72,&HC2,&H40,&H88,&HBE,&H94,&H76,&H9F,&HD1,&H66,&H50)
```

Výpočet X_A na kartě probíhá následujícím způsobem.

```
// Generace náhodného x_A jako string
xa$ = Hex$(Rnd) + Hex$(Rnd) + Hex$(Rnd)
// Uložení hodnoty generátoru do proměnné xxa$
xxa$ = G512N$
// Násobení xa$ · xxa$, výsledek uložen do xxa$
Call ECpMultiplyPoint (xxa$, xa$)
```

Nyní musí být APDU odpověď, ve které je vypočtený bod X_A zpracována v terminálu, kde se z obou souřadnic vytvoří bod na křivce typu ECPoint.

```
BigInteger Xax, Xay
// Odpověď APDU je uložena v proměnné „r“, x souřadnice je prvních
// 66B a y souřadnice je následujících 66B
Xax = new BigInteger(Arrays.copyOfRange(r.getBytes(), 0, 66));
```

```
Xay = new BigInteger(Arrays.copyOfRange(r.getBytes(), 66, 132));
ECPoint Xa = curve.createPoint(Xax, Xay);
```

Na konci druhé části protokolu, nazývané DH2Point, musí terminál spočítat:

$$H = x_B \cdot X_A,$$

$$\hat{\mathrm{G}} = sG + H.$$

```
// Výpočet bodu H
ECPoint H = Xa.multiply(new BigInteger(xb));
// Převedení dešifrovaného s z pole bajtů na string
String s = Hex.toHexString(decrypteds);
// Část násobení při výpočtu Ĝ, string s je v této chvíli číslo
// v decimální soustavě, karta je ovšem při násobení bodu na křivce
// chápe jako hexadecimální (a převádí před násobením na
// decimální), proto je stejný převod proveden i zde
ECPoint Gstr = G.multiply(new BigInteger(s,16));
// Část sčítání při výpočtu Ĝ
Gstr = Gstr.add(H);
```

Obdobně karta spočítá:

$$H = x_A \cdot X_B,$$

$$\hat{\mathrm{G}} = sG + H.$$

```
// Uložení X_B do dočasné proměnné Temp$
Temp$ = xxb$
// Násobením xa$ · Temp$ je získáno H, které zůstane uloženo v Temp$
Call ECpMultiplyPoint (Temp$, xa$)
// Do proměnné Gstr$ je předpřipraven generátor G
Gstr$ = G512N$
// Část násobení při výpočtu Ĝ
Call ECpMultiplyPoint (Gstr$, s$)
// Část sčítání při výpočtu Ĝ
Call EcpAddPoints(Gstr$,Temp$)
```

7.7 Třetí část PACE

Součástí třetí části je opět násobení bodu na eliptické křivce, obdoba DH protokolu na eliptických křivkách. Oproti předchozí kapitole je nahrazen generátor G vypočteným $\hat{\mathrm{G}}$.

Terminál vygeneruje náhodné y_B a spočte:

$$Y_B = y_B \cdot \hat{\mathrm{G}}.$$

Výsledek je odeslán na kartu, na kartě je vygenerováno y_A a spočítáno:

$$Y_A = y_A \cdot \hat{\mathrm{G}}.$$

Výsledek je naopak odeslán do terminálu, následně obě strany mohou vypočítat hodnotu K. Terminál:

$$K = y_B \cdot Y_A$$

a karta:

$$K = y_A \cdot Y_B.$$

7.8 Čtvrtá část PACE

V závěrečné části je z bodu K odvozen klíč pro šifrování (K_{enc}), podepisování (K_{mac}) i třetí klíč (K'_{mac}), který slouží k podepsání příslušného bodu Y a parametrů křivky $\mathcal{G}$. Jestliže podpisy souhlasí na obou stranách, pak byl PACE proveden správně. Všechny tři klíče jsou odvozeny pomocí hashovací funkce aplikované na bod K zřetězený s čísly 1, 2 a 3.

$$K_{enc} = \mathcal{H}(1||K),$$
$$K_{mac} = \mathcal{H}(2||K),$$
$$K'_{mac} = \mathcal{H}(3||K).$$

V terminálu se K_{enc} (SHA256 hash) spočte následovně.

```
// Uložení x a y souřadnice bodu K do stringu a zarovnání na 66B
String X = padding(K.normalize().getXCoord().toString(),132);
String Y = padding(K.normalize().getYCoord().toString(),132);
// Inicializace hashovací funkce
MessageDigest md;
md = MessageDigest.getInstance("SHA-256");
md.update(DatatypeConverter.parseHexBinary("01"));
md.update(DatatypeConverter.parseHexBinary(X));
md.update(DatatypeConverter.parseHexBinary(Y));
Byte[] Kenc = md.digest();
```

Následující hashe (klíče K_{mac} a K'_{mac}) jsou vytvářeny obdobně. Na kartě se hash tvoří funkcí Sha256Hash().

```
Kenc$ = Sha256Hash(Chr$(&H01) + K$)
```

Na konci PACE terminál spočte podpis T_B, podepisovanými daty jsou bod Y_A a parametry eliptické křivky $\mathcal{G}$.

$$T_B = MAC(K'_{mac}; (Y_A, \mathcal{G}))$$

Využijeme Bouncy Castle API, které nám poskytne možnost AES šifrování a CMAC podpisu.

```
Byte [] macTb = new byte[16];
// Inicializace CMAC podpisu, kde je použita symetrická šifra AES
CipherParameters params2 = new KeyParameter(Kmac2);
BlockCipher aes = new AESEngine();
CMac mac = new CMac(aes);
mac.init(params2);
// Přidání dříve uloženého body Ya
mac.update(YaTb, 0, 132);
// Přidání eliptické křivky, zde použitá P-512 je v Basic Card
// reprezentována číslem 19₁₀, což je 13₁₆.
mac.update(DatatypeConverter.parseHexBinary("0013"), 0, 2);
mac.doFinal(macTb, 0);
```

Vypočtený podpis je odeslán na kartu, která spočte vlastní T_B, pokud se shoduje s přijatým, je odpověď vracená v bajtech SW1 a SW2 rovna 9000, pokud se výsledky nerovnají, vrací karta hodnotu 9001.

Průběh výpočtu na kartě:

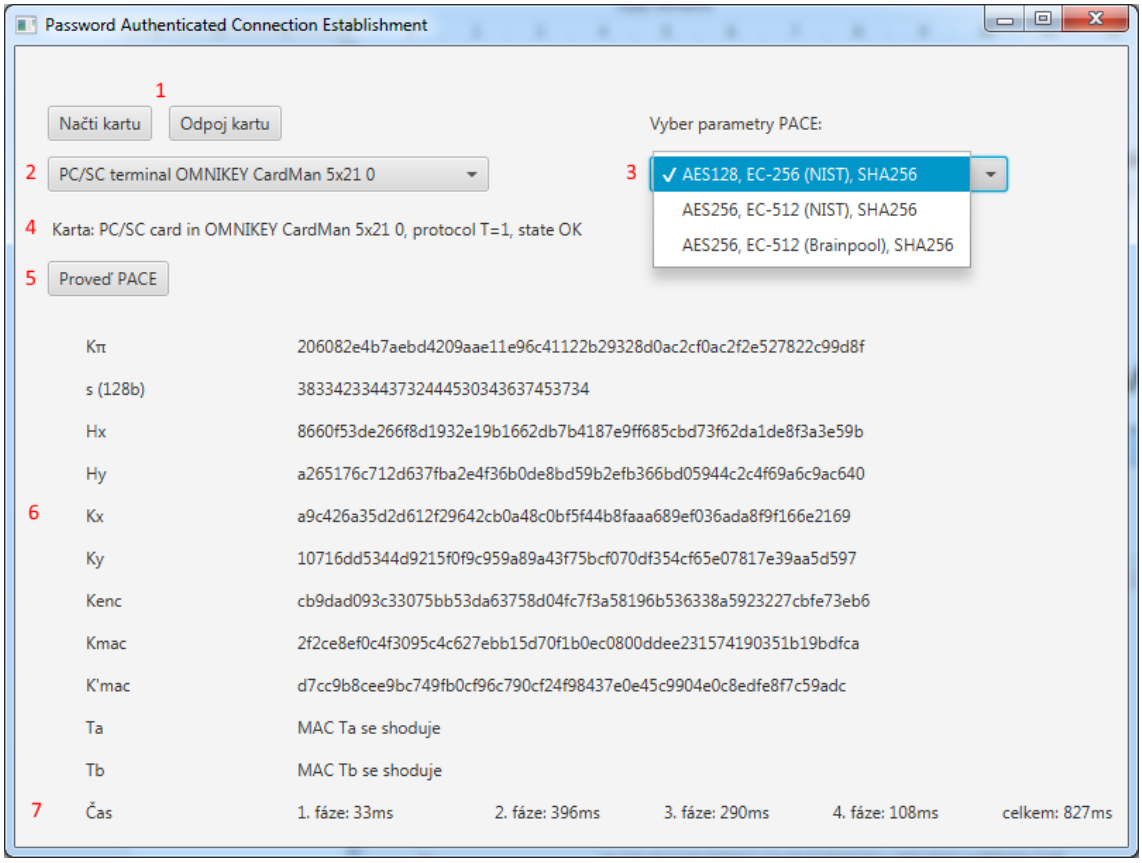
```
Command &H88 &H07 MACTb(Data$, Disable LE)
  Klic$ = Kmac2$
  // Uložení bodu Ya a čísla křivky do stringu (číslo křivky je nutné
  // převést z integeru na string)
  macTb$ = yya$ + CurveIndex% as String
  // Pro výpočet podpisu CMAC je použita funkce dostupná na kartě
  // nazvaná OMAC, vstupem je délka klíče AES šifry (256b), klíč
  // a string, který má být podepsán
  macTb$ = OMAC(256,Klic$,macTb$)
  // Nastavení bajtů SW1 a SW2
  If Data$ = macTb$ Then
    SW1SW2=&H9000
  Else SW1SW2=&H9001
End command
```

Výpočet T_A je obdobný, ovšem zde výpočet začíná karta. Terminál se poté na vypočtenou hodnotu dotáže a porovná se svojí. Podepisován je zde bod Y_B a opět parametry eliptické křivky $\mathcal{G}$.

$$T_A = MAC(K'_{mac}, (Y_B, \mathcal{G}))$$

8 POPIS GUI A OVLÁDÁNÍ APLIKACE

Aplikace s grafickým uživatelským rozhraním (viz Obr. 19) je vytvořena k provádění a testování protokolu PACE. Umožňuje zvolit a načíst libovolnou kartu platformy Basic card a také zvolit z několika sad kryptografických primitiv. Důležité hodnoty vypočtené v průběhu PACE jsou vypisovány do hlavního okna a dílčí části jsou měřeny z hlediska časové náročnosti.



Obr. 19: PACE – GUI

1. Tlačítka pro připojení, resp. odpojení karty. Toto je potřeba provést před prováděním PACE, jinak se objeví chybová hláška „Karta není připojena“.
2. Seznam všech čteček čipových karet aktuálně dostupných v systému Windows.
3. Zde lze vybrat sadu kryptografických primitiv, jsou zde tři volby:
    a. Doporučení NIST pro 128b bezpečnost – symetrická šifra AES-128, použitá eliptická křivka NIST P-256, použitá hashovací funkce SHA256 [26]
    b. Doporučení NIST pro 256b bezpečnost – symetrická šifra AES-256, použitá eliptická křivka NIST P-512, použitá hashovací funkce SHA256 [26]
    c. Obdoba předchozí varianty, pouze je eliptická křivka NIST nahrazena 512 bitovou křivkou Brainpool P512r1
4. Výpis připojené karty, název čtečky, kde je karta připojená a také použitý protokol, zde T=1.
5. Tlačítko pro spuštění provádění PACE protokolu
6. Výpisy všech podstatných hodnot při provádění protokolu. Hash K_π, náhodné číslo s (délka 128 bitů), obě souřadnice bodů K a H. Výsledné klíče K_{enc}, K_{mac} a K'_{mac}. Nakonec potvrzení, jestli se shodují nebo neshodují CMAC podpisy T_A a T_B.
7. Čas měřený ve čtyřech fázích protokolu, které jsou popsány v předchozích kapitolách.

9 ANALÝZA PACE

Všechny 4 fáze protokolu byly změřeny pro rozdílnou délku klíče u eliptických křivek a rozdílnou délku klíče AES šifry. Všechna měření byla prováděna 10 krát a zprůměrována. Časy jsou včetně komunikace karty s terminálem, počátek měření je zahájen stiskem tlačítka „Proveď PACE“ v terminálové aplikace a ukončen při ověření podpisů T_A a T_B. Měření je prováděno v terminálové aplikaci pomocí funkce *System.currentTimeMillis()*, která získá aktuální čas v milisekundách. Použita byla čtečka Omnikey CardMan 5x21 a kontaktní karta Basic card 7.6 rev D.

Tab. 8: Časy provádění PACE pro Basic card 7.6 rev. D

	AES128, NIST P-256	AES256, NIST P-512	AES256, Brainpool P512r1
První část PACE 1. Výpočet hashe na kartě i v terminálu. 2. Šifrování AES na kartě. 3. Dešifrování AES v terminálu.	32,7 ms	31,7 ms	31,9 ms
Druhá část PACE 1. Provedení ECDH algoritmu, zahrnuje na obou stranách generaci náhodného čísla a 2x násobení bodu eliptické křivky. 2. Výpočet $\hat{G}$ na obou stranách, nutnost násobení i sčítání na eliptické křivce.	386,8 ms	438,3 ms	462,7 ms
Třetí část PACE 1. Znovu provedení ECDH algoritmu, na rozdíl od předchozí fáze je zde použito $\hat{G}$ místo generátoru eliptické křivky G.	265,4 ms	299,5 ms	313,2 ms
Čtvrtá část PACE 1. Tvorba tří klíčů pomocí hashovací funkce. 2. Podpis CMAC a jeho následná kontrola.	102,1 ms	144,5 ms	142,8 ms
Celkem	787 ms	914 ms	950,6 ms

V první části Tab. 8 je patrné, že změna klíče u AES šifry neudělá téměř žádný rozdíl v rychlosti šifrování, hash zůstává pořád SHA256, zde se nic nemění.

Druhá část ukazuje, že použití delších klíčů u ECDH algoritmu a následných dalších operací na křivce zpomalí operaci o pouhých 12% v případě, že uvažujeme NIST křivky. Pokud porovnáme křivku NIST a Brainpool se stejnou délkou klíče (512b), je NIST křivka o 5% rychlejší. Tato vlastnost je způsobena tím, že NIST křivky používají Mersennova prvočísla, zatímco Brainpool křivky používají náhodná prvočísla [27].

Třetí část je velmi podobná druhé, jsou zde ovšem menší rozdíly v rychlostech, jelikož vynecháváme jednu operaci sčítání a jednu operaci násobení.

Poslední část ukazuje, že rychlost hashování záleží na délce hashovaných dat, proto při hashování bodu s 256b délkou souřadnic je rychlejší, než hashování bodů s 512b délkou.

9.1 Srovnání různých verzí Basic card

Klíčovou knihovnu EC-p pro operace nad eliptickými křivkami dle dokumentace podporují karty řady ZC7 a ZC8. Z těchto řad byly k dispozici karty:

- ZC 7.5 rev. B
- ZC 7.6 rev. D
- ZC 8.6 rev. D

Během měření bylo zjištěno, že kata ZC 8.6 podporuje AES pouze s délkou klíče 128b, proto všechna měření byla prováděna s použitím AES128 a křivkou NIST P-512. Karta ZC 7.5 i přes opačné tvrzení výrobce knihovnu EC-p nepodporuje a tudíž musela být z měření odstraněna.

Karta ZC 8.6 je tzv. MultiApplication card, takže umožňuje instalaci více aplikací zároveň. V našem případě stačí použít jednu aplikaci i přesto je potřeba ji na kartě nastavit jako výchozí, což provedeme následujícím blokem kódu.

```
#Pragma DefaultApp("\DefaultApp")
#Include COMPONNT.DEF
  Dir "\"
    Application "DefaultApp"
      Lock=Execute:Always
  End Dir
```

Výsledný naměřený čas je vždy průměrem z 20 hodnot.

Tab. 9: Srovnání karet ZC 7.6 a ZC 8.6

	ZC 7.6 rev. D	ZC 8.6 rev. D
AES128, NIST P-512	908,5 ms	915,2 ms

Z měření vidíme (viz Tab. 9), že rozdíly mezi kartami jsou minimální. Mírně delší časy u karty ZC8.6 mohou být způsobeny vyšší složitostí multiaplikačních karet, kdy je potřeba nejprve zvolit správnou aplikaci a poté vykonat program.

9.2 Bezpečnostní analýza PACE

Protokol PACE byl analyzován bezpečnostními experty z několika úhlů pohledu. V roce 2009 byla provedena analýza kryptografické bezpečnosti výzkumníky z Technické univerzity Darmstadt. [25]

Při jednoduchém ustanovení klíče založeném na předsdíleném hesle s nízkou entropií, je veškerá bezpečnost závislá pouze na síle tohoto hesla. Heslo se může skládat jenom z určitých znaků a může mít pouze omezenou délku, pak počet možných hesel označíme N. Heslo lze uhodnout s minimální pravděpodobností $1/N$. Pokud by se heslo přímo použilo k šifrování komunikace, mohl by útočník použít slovníkový útok, čímž by výrazně zvýšil pravděpodobnost nalezení hesla. Protokoly pro ustanovení klíče na základě předsdíleného hesla se snaží, aby pravděpodobnost jeho nalezení byla co nejblíže $1/N$.

K testování bezpečnosti PAKE protokolů lze využít model popsaný v [28], který byl použit i pro analýzu PACE. Dále bylo využito náhodné orákulum i model ideální šifry. Analýza prokázala, že PACE lze považovat za bezpečný.

Druhá nezávislá analýza zkoumala bezpečnost protokolu z hlediska algebraické logiky [29] a opět nebyly nalezeny žádné zranitelnosti, které by bránily využití PACE.

Jelikož je PACE založen na několika krocích, případné prolomení jednoho z nich obvykle nezpůsobí narušení bezpečnosti celého protokolu. V prvním kroku lze útočit na kryptogram z. Pokud by útočník hádal hesla π a počítal z nich hashe, kterými by se snažil dešifrovat kryptogram, bylo by pro něj velmi obtížné rozpoznat, jestli použil správný klíč, protože zašifrovaná hodnota je náhodně generovaná.

Narušení druhého kroku také nenaruší bezpečnost protokolu, protože i kdyby se podařilo útočníkovi získat výslednou hodnotu $\hat{G}$ pořád by musel vyřešit problém diskrétního logaritmu na eliptických křivkách v kroku následujícím.

9.3 Návrhy na zlepšení PACE

I přes provedené analýzy, které dokazují bezpečnost protokolu, je možné použitím správných kryptografických primitiv a jejich správnou implementací bezpečnost zvýšit, případně zvýšit rychlost provádění protokolu.

Několikrát v průběhu PACE je prováděno hashování. Alespoň v prvním kroku protokolu, kdy se hashuje přímo předsdílené heslo by bylo vhodné použít techniku solení. Tato technika by zabránila útoku na hash pomocí předem generovaných hashů (např. formou rainbow tables). Dále lze uvažovat o použití silnější hashovací funkce SHA3. Platforma Basic card zatím nemá SHA3 implementovanou, a tudíž v této práci nebyla použita.

Z měření vyplývá, že nejpomalejší část protokolu je násobení bodů na eliptických křivkách. Na každé straně dochází 5 krát k násobení bodu na křivce, což především na limitovaném hardwaru čipové karty trvá nezanedbatelnou dobu. Použití novějších eliptických křivek by mohlo zaručit vyšší bezpečnost a zároveň vyšší rychlost násobení. Jednou z možností jsou Edwardsovy křivky [30], zkoumané roku 2007. Basic card umožňuje použití vlastních křivek, jedná se ovšem o velmi málo zdokumentovanou funkci a rovněž manuál navrhuje obrátit se s požadavkem přímo na výrobce karty.

Jedním z dalších zlepšení je i přidání aktivní autentizace. PACE dokáže ověřit, že na kartě je uložené správné heslo π a na základě toho předpokládá, že se jedná o vlastníka karty. Nedokáže už ale určit, že se jedná o originální čip. Každý čip, který zná správné heslo dokáže ustanovit klíč pomocí PACE. Aktivní autentizace probíhá tak, že karta zašle

terminálu zašifrovaný certifikát, terminál jej dešifruje a ověří pravost čipu. Tento proces lze dále zjednodušit, pro aktivní autentizaci se využije část násobení na eliptických křivkách z předchozích kroků PACE [31].

10 ZÁVĚR

Zkoumané operační systémy čipových karet nabízí všechny důležité kryptografické funkce, jako jsou hashovací funkce a operace symetrické a asymetrické kryptografie. Největší množství kryptografických funkcí nabízí platforma Java card, ovšem ne každý výrobce všechny funkce implementuje a získat čipové karty s nejnovější verzí platformy je často obtížné. Nejjednodušší vývoj je u karet Basic card, kde je k dispozici zdařilé vývojové prostředí a lze snadno vytvářet zároveň terminálovou aplikaci i aplikaci na kartě.

Měření rychlosti komunikace mezi čipovou kartou a terminálem ukázalo, že v mnoha případech je rychlejší bezkontaktní protokol T=CL oproti kontaktním protokolům T=0 a T=1. U bezkontaktního přenosu je ovšem vyšší riziko odposlechů a je proto nutné využít vhodný protokol pro ustanovení klíče.

V práci jsou představeny různé varianty těchto protokolů a jejich případné implementace na čipových kartách.

Následně byl vypracován popis protokolu PACE, návrh implementace na platformě Basic card a následná implementace. Terminálová aplikace je naprogramována v jazyce Java a s čipovou kartou komunikuje pomocí APDU příkazů, celá komunikace mezi terminálovou aplikací a aplikací na čipové kartě je v práci detailně popsána.

Výsledné měření protokolu PACE ukazuje, že nejvíce časově náročná operace je násobení bodu na eliptické křivce. Volbou eliptických křivek tudíž lze výrazně ovlivnit rychlost protokolu. S křivkou NIST P-256 a čipovou kartou Basic card 7.6 rev. D trvalo provedení protokolu 787 ms. Při použití křivky NIST P-521 se čas zvýšil na 914 ms a u křivky Brainpool P512r1 na 951 ms, což dokazuje vyšší rychlost NIST křivek. Při srovnání dvou verzí Basic card, konkrétně 7.6 rev. D a 8.6 rev. D byla zjištěna mírně vyšší rychlost protokolu na kartě verze 7.6, což je dáno pravděpodobně jednodušší softwarovou architekturou karty, jelikož karta 8.6 je tzv. multiaplikační.

Práce ukazuje, že protokoly PAKE (zejména testovaný PACE) je možné implementovat na soudobé čipové karty a s patřičnými optimalizacemi lze dosáhnout časů pod 1 s, což je přijatelný čas pro většinu oblastí použití (např. přístupové a platební systémy). K dosažení přijatelných časů je nutná přímá podpora potřebných kryptografických funkcí na čipové kartě. Vlastní softwarová implementace náročných matematických výpočtů totiž významně zpomalí celý protokol.

LITERATURA

[1] RANKL, Wolfgang. *Smart Card Applications: Design Models for using and programming smart cards*. Chichester, West Sussex: John Wiley & Sons Ltd., 2007. ISBN 978-0-0470-05882-4.

[2] *Java Card 2.2 Runtime Environment (JCRE) Specification*. Palo Alto, CA, 2002.

[3] Java Card 3.0.5 is now available. *Oracle* [online]. 2015 [cit. 2017-05-09]. Dostupné z: https://blogs.oracle.com/ericv/entry/java_card_3_0_5

[4] *The Java Card 3 Platform*. Santa Clara, CA: Sun Microsystems, 2008.

[5] VOMÁČKA, Bc. Martin. *Moderní přístupový systém*. Brno, 2016. Diplomová práce. VUT. Vedoucí práce Ing. Lukáš Malina, Ph.D.

[6] ORTIZ, C. Enrique. *An Introduction to Java Card Technology* [online]. 2003 [cit. 2016-10-27]. Dostupné z: http://www.oracle.com/technetwork/java/javacard/javacard1-139251.html

[7] GUILFOYLE, Tony. *Basic Card Developers Guide*. 8.15. Minden, Germany, 2012. Dostupné také z: http://209.68.36.204/downloads/bc_pdf.zip

[8] ŠVENDA, Petr. Internal format of .NET smart card communication. *Fakulta informatiky - Masarykova Univerzita* [online]. Brno, 2009 [cit. 2016-12-14]. Dostupné z: http://www.fi.muni.cz/~xsvenda/dotnetcard.html

[9] KOČÍŘ, Michal. *Použití Smart-karet v moderní kryptografii*. Brno, 2013. Diplomová práce. VUT. Vedoucí práce Ing. Jan Hajný, Ph.D.

[10] *MULTOS Developer's Guide*. Londýn, Spojené krátlovství: MAOSCO Limited, 2016.

[11] *IDPrime .NET Smart Card*. D1265202C. Francie: Gemalto, 2014.

[12] *Java Card 3 Platform - Developers Guide*. 3.0.5. Oracle, 2015.

[13] BELLOVIN, Steven M. a Michael MERRITT. *Encrypted Key Exchange: Password-Based Protocols Secure Against Cidtionary Attacks*. Murray Hill, NJ, 1992.

[14] JABLON, David P. *Strong Password-Only Authenticated Key Exchange*. Westboro, MA, 1996.

[15] HAO, Feng a Siamak F. SHAHANDASHTI. *The SPEKE Protocol Revisited*. Newcastle University, Spojené království.

[16] ZHANG, Muxiang. Analysis of the SPEKE password-authenticated key exchange protocol. Waltham, USA: IEEE, 2004. ISSN 1089-7798.

[17] HAO, Feng a Peter RYAN. *Password Authenticated Key Exchange by Juggling*. Center for Computational Science, Londýn, 2008.

[18] GREEN, Matthew. Zero Knowledge Proofs: An illustrated primer. *Cryptography Engineering* [online]. 2014 [cit. 2017-05-09]. Dostupné z: https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/

[19] Schnorr Signature. *Git Books* [online]. [cit. 2017-05-09]. Dostupné z: https://chunminchang.gitbooks.io/j-pake-over-tls/content/appendix/zkp/schnorr_signature.html

[20] PERLMAN, Radia a Charlie KAUFMAN. *PDM: Password-Derived Moduli*. Washington, D.C., USA, 2011.

[21] Autentizační protokoly založené na heslech a jejich implementace v prostředí čipových karet. Brno, 2008. Diplomová práce. MU. Vedoucí práce Mgr. et Mgr. Jan Krhovják.

[22] ITOI, Naomaru, Tomoko FUKUZAWA a Peter HONEYMAN. *Secure Internet Smartcards*. University of Michigan.

[23] HOLZL, Michael, René MAYRHOFER, Endalkachew ASNAKE a Michael ROLAND. A Password-authenticated Secure Channel for App to Java Card Applet Communication. Rakousko.

[24] *Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token*. 2.20. Bonn, Germany, 2015.

[25] BERDEN, Jens, Marc FISCHLIN a Dennis KUGLER. *Security Analysis of the PACE Key-Agreement Protocol*. Germany, 2009.

[26] GIRY, Damien. Cryptographic Key Length Recommendation. *BlueKrypt* [online]. 2017 [cit. 2017-04-26]. Dostupné z: https://www.keylength.com/en/4/

[27] BAKKER, Paul. Elliptic Curve performance: NIST vs Brainpool. *ARMmbed* [online]. 2013 [cit. 2017-04-26]. Dostupné z: https://tls.mbed.org/kb/cryptography/elliptic-curve-performance-nist-vs-brainpool

[28] ABDALLA, Michel, Pierre-Alain FOUQUE a David POINTCHEVAL. *Password-Based Authenticated Key Exchange in the Three-Party Setting*. Paříž, Francie, 2005.

[29] FISCHLIN, Marc, Lassaad CHEIKHROUHOU, Werner STEPHAN, Ozgur DAGDELEN a Markus ULLMANN. *Merging the Cryptographic Security Analysis and the Algebraic-Logic Security Proof of PACE*. German Research Center for Artificial Intelligence, 2011.

[30] BOS, Joppe W., Craig COSTELLO, Patrick LONGA a Michael NAEHRIG. Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis. 2014.

[31] HANZLIK, Lucjan, Łukasz KRZYWIECKI a Mirosław KUTYŁOWSKI. *Simplified PACE|AA protocol*. Vratislav, Polsko, 2013.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

A	Doba přenosu APDU [ms]
a,b,x	Náhodně volená čísla
C	Kryptogram
$D()$	Transformace dešifrování
$E()$	Transformace šifrování
g	Generátor
h	Kofaktor
$H()$	Transformace hashování
K	Klíč
n	Řád generátoru
p	Velké prvočíslo
R	Přenosová rychlost [bps]
SK	Soukromý klíč
V_{cc}	Napájecí napětí
VK	Veřejný klíč
π	Předsdílené heslo

AEAD	Authenticated Encryption with Associated Data, autentizované šifrování s asociovanými daty
AES	Advanced Encryption Standard, standard pokročilého šifrování
AID	Application Identifier, identifikátor aplikace
APDU	Application Protocol Data Unit, datová jednotka aplikačního protokolu
API	Application Programming Interface, rozhraní pro programování aplikací
ATR	Answer to Reset, odpověď na resetování
AUX	Auxiliary, pomocný
BSI	Bundesamt für Sicherheit in der Informationstechnik, Spolkový úřad pro bezpečnost informační techniky
CCM	Counter with Cipher Block Chaining MAC
CIL	Common Intermediate Language
CLA	Class, třída

CLK	Clock, hodinový signál
CLR	Common Language Runtime
CMAC	Cipher-based Message Authentication Code
CPU	Central Processing Unit, centrální procesorová jednotka
CRC	Cyclic Redundancy Check, cyklický redundantní součet
DES	Data Encryption Standard, standart pro šifrování dat
DF	Dedicated File
DH	Diffie-Hellman
DLL	Dynamic-link Library, dynamicky linkovaná knihovna
DVB-C	Digital Video Broadcast – Cable, kabelové digitální televizní vysílání
DVB-S	Digital Video Broadcast – Satellite, satelitní digitální televizní vysílání
EC	Elliptic Curves, eliptické křivky
ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography, kryptografie na eliptických křivkách
ECDH	Elliptic Curves Diffie-Hellman
ECNR	Elliptic Curves Nyberg–Rueppel
EC-SRP	Elliptic Curves Secure Remote Password
EEPROM	Electrically Erasable Programmable Read Only Memory, elektronicky vymazatelná paměť pouze pro čtení
EF	Elementary File
EKE	Encrypted Key Exchange, šifrovaná výměna klíčů
GCM	Galois Counter Mode
GND	Ground, uzemnění
GP	Global Platform
GUI	Graphical User Interface, grafické uživatelské rozhraní
HMAC	Keyed-hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output, vstup/výstup
ID	Identifikátor
IEC	International Electrotechnical Commission, Mezinárodní elektrotechnická komise

IEEE	Institute of Electrical and Electronics Engineers, Institut pro elektrotechnické a elektronické inženýrství
INS	Instruction, instrukce
IP	Internet Protocol
ISO	International Organization for Standardization, Mezinárodní organizace pro normalizaci
JCOP	Java Card OpenPlatform
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
J-PAKE	Password Authenticated Key Exchange by Juggling
JRE	Java Runtime Environment
JVM	Java Virtual Machine
L_c	Length command, délka příkazu
L_e	Length expected, očekávaná délka odpovědi
MAC	Message Authentication Code
MD5	Message Digest 5
MEL	MULTOS Executable Language
MF	Master File
OMAC	One-key Message Authentication Code
OS	Operační systém
PACE	Password Authentication Connection Establishment
PAKE	Password Authenticated Key Exchange
PDM	Password Derived Moduli
PIN	Personal Identification Number, osobní identifikační číslo
PPS	Protocol Parameter Selection
PRNG	Pseudo Random Number Generator, generátor pseudo-náhodných čísel
ROM	Read Only Memory, paměť pouze pro čtení
RSA	Rivest, Shamir, Adleman
RST	Reset
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module, účastnická identifikační karta
SPEKE	Simple Password Exponential Key Exchange
SPU	Standard or proprietary use, standartní nebo proprietární použití
SRP	Secure Remote Password

TPDU	Transport Protocol Data Unit, datová jednotka transportního protokolu
TRNG	True Random Number Generator, generátor náhodných čísel
URI	Uniform Resource Identifikator, jednotný identifikátor zdroje
USB	Universal Serial Bus, univerzální sériová sběrnice
ZC	ZeitControl
ZKP	Zero-knowledge Proof, důkaz s nulovou znalostí

SEZNAM PŘÍLOH

A IMPLEMENTACE PROTOKOLU PACE

Příloha je přiložena na CD. Obsah CD:

Aplikace,

Zdrojovy kod,

readme.txt.